(2)

RADC-TR-89-103
Final Technical Report
August 1989

VHLL SYSTEM PROTOTYPING TOOL

International Software Systems, Inc.

Don Hartman, Mike Konrad, Terry Welch

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-89-103
6a. NAME OF PERFORMING ORGANIZATION International Software Systems, Inc.	6b. OFFICE SYMBOL (If applicable) N/A
6c. ADDRESS (City, State, and ZIP Code) 9490 Research Blvd, Suite 200 Austin TX 78759	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COEE) 7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center	8b. OFFICE SYMBOL (If applicable) COEE
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0129 10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
62702F	5581	22	17

11. TITLE (Include Security Classification)

VHLL SYSTEM PROTOTYPING TOOL

12. PERSONAL AUTHOR(S)
Don Hartman, Mike Konrad, Terry Welch

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Jun 85 TO Sep 88	August 1989	64

16. SUPPLEMENTARY NOTATION

N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
12	05		Functional Specifications, Executable Specifications, Reusability, Requirements Analysis

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report contains the results of studying issues relating to the requirements development process. The goals, objectives, results and conclusions for three separate but related components, 1) a Requirements Engineering Panel study, 2) the development of a Very High Level Language Prototyping Tool, and 3) a "Database Management Study" are described.

First, a panel of experts was formed whose goals were to understand the requirements engineering problem and define a long-range (10 year) Research and Development plan for the RADC Requirements Engineering Testbed (RET). The plan was to identify tools and methods that should be developed, evaluated and integrated.

Second, the objective of the rapid prototyping system, "Proto", was to develop a functional prototyping capability, in which a systems analyst could define and validate a functional specification, prior to extensive design and coding effort in a large (C3I)-(see reverse)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL William E. Rzepka	22b. TELEPHONE (Include Area Code) (315) 330-2762	22c. OFFICE SYMBOL RADC (COEE)

DD Form 1473, JUN 86 *Previous editions are obsolete.*

Block 19 - (continued)

software system. The approach includes defining a functional specification language for defining prototypes, developing and organizing a facility around the concept of reusability with the reusable components (both functions and data types) managed by an object manager, and develop a set of support tools to facilitate the construction and execution of prototypes.

Finally, a study was performed to identify the data base management requirements for supporting an automated requirements engineering process. That is, specify a DRMS that will serve as an RET integration vehicle for selected tools permitting an analyst to share requirements data between tools. The approach includes identifying database issues in software requirements development, prototyping a DBMS, and specifying the kind of DBMS needed to tightly couple requirements tools in the RET.

Table of Contents

EXECUTIVE SUMMARY

The overall goal of this contract was to analyze the requirements development process then define new methods and develop prototypes of tools to improve this process.

The contract included three separate but related components: 1) a Requirements Engineering Panel study; 2) the development of a Very High Level Language Prototyping Tool; and 3) a Database Management Study". This report describes the goals, objectives, results and conclusions reached for each of these three components.

Requirements Engineering Panel

A panel of experts was formed whose goals were to understand the requirements engineering problem and define a long-range (10 year) Research and Development plan for the RADC Requirements Engineering Testbed (RET). The plan was to identify tools and methods that should be developed, evaluated, and integrated.

The panel met several times and produced a technical report entitled "RADC Requirements Engineering Testbed Research and Development Program Panel Recommendations" [2]. The Panel recommended that RADC pursue a research and development program for the RET consisting of two tracks:

Evolutionary Track

This track proposed an evolutionary R&D effort to extend the current formalisms and tools. Initial efforts are toward the development of tools for prototyping interfaces and functionality, and in deriving performance estimates based on estimated or simulated work loads. Future efforts would develop tools and methods that aid in: (1) scenario development, analysis, and execution; (2) cost, risk, and performance analysis; (3) the acquisition, modeling, and usage of domain information; and (4) requirements analysis methodology.

Formal Language Track

This track proposed that research effort be spent toward developing a single formal language for expression of goals, requirements, and solution architectures. With this language, users could automatically generate prototypes from a formal specification.

The panel further recommended a testbed integration plan for 1990, that would lead to a uniform user interface to all tools and a common repository for all requirements, designs, and tool data, and which incorporates RADC's currently-contracted requirements tools.

Very High Level Language Prototyping Tool

A rapid prototyping system called "Proto" was developed to support prototyping of C3I applications.

The objective of the Proto effort was to develop a functional prototyping capability, in which a systems analyst could define and validate a functional specification, prior to extensive design and coding effort in a large software system.

The approach taken in attempting to realize this objective included:

a) Define a functional specification language for defining prototypes.

b) Organize the facility around the concept of reusability with the reusable components (both functions and data types) managed by an object manager.

c) Develop a set of support tools to facilitate the construction and execution of prototypes.

The functional specification language was developed and used to construct several prototype examples. The language has a graphical data flow-like syntax with hierarchical decomposition capabilities. Experience indicates that this type of language is a good approach for a functional specification language. It is natural to use and provides a good communications medium between analysts and users of target systems. Additionally, it properly focuses the analyst on abstract datatypes and functions early in the specification process, and discourages users from jumping into design level issues early in the process. The ability to execute the specification to determine how well the prototype meets the functional requirements is highly desirable.

The Proto language has precise semantics and a very simple syntax. This has proven to have both advantages and disadvantages. On the plus side, users can learn to use the language quickly and do not have to worry about the execution order of the components in a diagram - namely, the scheduling is completely automatic. On the negative side, the fact that Proto does not allow the user to directly control scheduling results in situations when the Interpreter does not execute complex diagrams in the order that the user might consider intuitive.

The Proto system centers around the concept of building prototypes from reusable components. As part of the work, a small library of reusable components was constructed and used to build up several sample prototype systems. The library components are both domain dependent (e.g., algorithms associated with C3I), and domain independent (e.g., user interface prototyping tools). Our preliminary conclusions drawn from these examples are that 1) a user can build a prototype quickly if the reusable library contains most of the functions required for that particular example, and 2) a lot more work on reusability must be done before a analyst who is not an expert in using Proto will be able to quickly put together a prototype.

The support environment that was developed is highly graphical and includes an object-oriented interface supported by an object manager. With this interface, the user points at the object of interest and sees a menu of options that are meaningful to the selected object in the current context (e.g., Editor versus Interpreter).

The Proto support tools that were developed included: 1) a diagram Editor through which users can construct executable functional specifications; 2) an Interpreter that provides for direct execution of Proto diagrams (graphs); and 3) a Browser which allows a user to navigate through the database of objects.

The Editor tool understands the syntax of a diagram and supports the user by not allowing meaningless diagrams to be created. The Editor provides a number of advanced editing features to simplify the task of the user. As might be expected experience has shown that the Editor could be improved. Some of the more desirable

features not yet implemented include a flexible cut-and-paste facility, the ability to split and merge multi-level graphs, and user definable icons.

The Proto Interpreter has worked quite nicely for the examples built to date. It provides the ability to watch the functional prototype run (e.g., the bubble(s) currently executing are highlighted), the ability to set graphical breakpoints, and the ability to instrument diagrams (i.e., display the values of selected data items along the arcs over which the data flows). Its current limitations include the fact that it is single threaded and has a fixed scheduling algorithm.

The existing implementation of the Proto Browser provides very basic capabilities for viewing and and modifying objects in the reusable library. While the current capability is adequate for the prototype, a more integrated graphical browser should be developed.

Database Management Study

A study was performed to identify the database management requirements for supporting an automated requirements engineering process.

The objective of the DBMS effort was to specify a DBMS that will serve as an RET integration vehicle for selected tools, in particular the tools that constitute the initial RET requirement capability, permitting an analyst to share requirements data between tools.

The approach taken in the DBMS effort was to:

(1) identify database issues in software requirements development, characterizing requirements data and data access patterns (usage of requirements data),

(2) from the analysis in (1), prototype a DBMS that would serve as the Proto database, coupling the Proto tools, permitting a sharing of requirements data across Proto tools, and

(3) on the basis of experience gained from the Proto DBMS effort, specify the kind of DBMS needed to tightly couple requirements tools in the RET.

Based on the Database Management study and the results of the Proto development effort, our conclusion is that an Object Manager should be used to manage all requirements engineering information in the RET. This model will provide a natural way to define the artifacts for representing the information required by the requirements engineering process and will provide the optimal way for managing the complex relationships among these artifacts. Other models (e.g., the relational, hierarchical or network models) could be used but they would suffer both in terms of performance and ease of use as compared to the object model.

SECTION 1. INTRODUCTION

The overall goal of this contract was to analyze the requirements development process then define new methods and develop prototypes of tools to improve this process.

1.1 Requirements Terminology

Requirements are precise statements of need which characterize a needed system in terms of its external characteristics (especially interfaces and functionality visible to the user) and constraints (e.g., on execution performance and reliability, and on development cost and time).

Requirements engineering is the systematic application of tools and techniques to guide and control the emergence of the requirements product. It is an iterative process of analysis, specification, evaluation, and refinement. Its inputs are mission-related ideas and problems expressed by mission specialists and their representatives. Its output is the requirements specification.

1.2 The Requirements Challenge

The current state of the requirements process is that it is almost entirely a manual process whose success depends on the insight of the analyst doing the work. There are no generally accepted methodologies, criteria for quality, or notations.

Typically requirements are almost entirely English text, so they are subject to differing interpretations by the customer (end-user) and by developers, namely the two audiences that the requirements statements should bring together.

Existing requirements techniques, such as SREM and PSL/PSA, provide a discipline for defining requirements and analyzing them, but are best at addressing late "requirements phase" concerns. They require a largely manual execution of the requirements process. It is not yet evident that the labor involved in their use is justified by the improved results obtained.

Opportunities for improving the requirements development process include:

(1) formal descriptions - improve precision in requirements statements, reduce ambiguity, and provide greater opportunity for machine-assisted analysis;

(2) domain information - capture and interpretation of context information will lead to improved requirements mechanization and interpretation;

(3) prototypes - end-users are better able to analyze their needs when they see operational results;

(4) scenarios - end-user needs are often expressed and analyzed in terms of scenarios; and

(5) cost and risk estimation - identifying and quantifying costs and risks in system development and operation is the basis for making informed trade-offs against system functionality, usability, and performance.

Each of these areas is presently very labor intensive and/or lacks a formal basis.

1.3 The Contract

The contract included three separate but related components:

Requirements Engineering Panel

A panel of experts was formed whose goals were to understand requirements engineering problems and develop a process model.

The panel met several times and produced a technical report entitled "RADC Requirements Engineering Testbed Research and Development Program Panel Recommendations" [2]. This report includes a plan for the development of a Requirements Engineering Testbed of tools and methodologies for defining requirements.

Very High Level Language Prototyping Tools

A rapid prototyping tool called "Proto" was developed to support prototyping of C3I applications.

This tool allows users to quickly create an executable prototype of C3I systems using a dataflow-like graphical language. The "VHLL System Prototyping Tool Users Manual" [4] describes this tool.

Database Management Study

A study was performed to identify the database management requirements for supporting an automated requirements engineering process.

The result of this study is a report entitled "A Data Management Facility for Requirements Engineering Testbed (RET)" [8].

The following report describes the objectives, approach, and results for each of the three components of the contract. It also identifies key conclusions reached as a result of the work.

1.4 Terms and Abbreviations

ADI	Atmospheric Defense Initiative
ASE	Advanced Sensor Exploitation
DBMS	Data Base Management System
C3I	Command, Control, Communications and Intelligence
OM	Object Manager
Proto	ISSI's rapid prototyping tool
PSL/PSA	Problem Statement Language / Problem Statement Analyzer - a requirements tool
RADC	Rome Air Development Center
R&D	Research and Development
RET	Requirements Engineering Testbed
RPS	Rapid Prototyping System - developed by Martin Marietta
SREM	Software Requirements Engineering Methodology
VHLL	Very High Level Language

SECTION 2. OBJECTIVES

RADC has a long-term goal of developing and evaluating new tools and methods that will enable Air Force systems analysts to address their requirements engineering problems. This contract's objectives address different aspects of this RADC goal, and thus it is appropriate to discuss what RADC proposes to do toward this goal before characterizing the contract objectives.

2.1 RADC Objectives - the Requirements Engineering Testbed (RET)

New requirements tools and methods (perhaps addressing one or more of the opportunities identified in Section 1.2) need to be evaluated on realistic problems in an environment that supports their use by Air Force systems analysts. Therefore, RADC proposes the creation of a facility which promotes experimentation, called the Requirements engineering Testbed (RET), and which hosts the new methods and tools. Air Force systems analysts will use the RET to define, analyze, and exercise requirements of planned operational systems. This will have two benefits: (1) new tools and methods will have early exposure on realistic problems, encouraging their use by the Air Force and industry and (2) evaluation of the new tools and methods in terms of requirements quality and productivity will provide insight for the next generation of tools.

In the near term, the RET will host three tools recently developed through RADC contracts. These tools will constitute the initial RET requirements capability:

(1) the **Analyst** developed by Systems Designers under subcontract to Imperial College, assists in organizing the requirements and context information by end-user role, aids in the documentation of major usage scenarios and performance and reliability stress points, and provides a capability to symbolically animate the requirements;

(2) the VHLL System Prototyping Tool, or **Proto**, - developed by International Software Systems, permits rapid definition of functional prototypes via a VHLL that is used to specify the functionality of the target system; the resulting description is executable and can be exercised by end-users, giving the opportunity to explore their needs in actual usage; and

(3) the **Rapid Prototyping System** (RPS), developed by Martin Marietta, permits rapid definition of graphic and dynamic user displays that can be driven, say, by a battle scenario; and provides the capability to define and analyze the results of several performance simulations: of an end-user workstation, of the system as a whole, and of processor and communication network resources.

RADC's objectives for the RET are further elaborated in [2]. The tools that constitute the initial RET requirements capability are additionally characterized in [3], [4], and [5].

2.2 Contract Objectives

With the RET described, we can now describe the objectives of each of the three parts of the contract.

2.2.1 Panel Objectives

The objective of the Panel of experts was to define a long-range (10 year) Research and Development (R&D) plan for the RET. This plan was to identify tools and methods that should be developed, evaluated, and integrated. The Panel was to be a broad-based effort with inputs from both academia and industry.

2.2.2 Proto Objectives

The objective of the Proto effort was to develop a functional prototyping capability, in which a systems analyst could define and validate a functional specification, prior to extensive design and coding effort in a large software system.

The principal benefit in a validated functional specification is the elimination of a major source of costly errors: poorly-understood end-user needs. If the functional specification correctly specifies end-user functionality, and if the developers can correctly develop a system to satisfy that specification, then there is a much greater chance that the resulting system will in fact behave as desired.

In order for this benefit to be realized a functional specification should be:

1) precise - so that end-users and software developers have a common understanding of the functionality to be delivered (i.e., what responses the system should provide to what inputs),
2) validatable - so that it can be determined whether the functions defined within the specification are in fact the "correct" functions an end-user needs in the target context, and
3) specifiable with modest effort - so that there are sufficient project resources and time to pursue system design and development.

2.2.3 DBMS Objectives

The objective of the DBMS effort was to specify a DBMS that will serve as an RET integration vehicle for selected tools, in particular the tools that constitute the initial RET requirement capability, permitting an analyst to share requirements data between tools.

SECTION 3. APPROACH

In this section, we explain the approaches taken toward satisfying the three contract objectives described in Section 2.2.

3.1 The Requirements R&D Plan Effort

As indicated earlier, a panel from academia and industry was formed to define a long-term R&D plan for introducing new methods and tools into the RET and for aiding in their evaluation. The panel was appointed and convened several times during mid-1985 to early 1986.

The panel's approach at defining the R&D plan consisted of:

(1) characterizing requirements issues from perspectives of mission specialists (end-users), acquisition engineers, and system developers - to more fully understand the requirements problem;
(2) defining a requirements engineering process model as a means of capturing the panel's understanding of the requirements engineering process, and as a basis for

systematic analysis of requirements issues and technology options;

(3) creating two scenarios illustrating the process model in terms of RET capabilities, one for 1990 and the other for 1995; and

(4) defining an RET R&D program as the strategy to provide the tools and methods that would work together in the ways envisioned in the process model and in the 1990 and 1995 scenarios.

The result of the panel's efforts is described in a report [2]. A summary of the panel's findings is presented in Section 4.1.

3.2 The VHLL System Prototyping Tools Effort

The objective of the Proto effort was to develop a functional prototyping capability. Some of the key ideas behind the approach taken in the design include:

1) The prototyping facility should provide a precise and simple modeling capability (language) so that users and developers will have a common understanding of the functionality provided by a model.

2) The facility should make it simple for developers to define the solution architecture (specification) and to determine that the architecture meets the system requirements.

3) Prototypes developed on the facility should be executable with the goals of a) assisting users to understand the capabilities and limitations of the target system; and b) assisting users in validating the system requirements.

4) Developing a prototype using the facility must take no more than a small fraction of the time required to develop the target system using conventional development technology.

5) The facility should be useful for developing scenarios of usage and test patterns that can later be used for testing the target system.

6) The facility should provide prototype user interfaces that allow users to evaluate the target system's usability.

7) The facility should be helpful in checking the interoperability of subsystems via the interoperation of prototypes.

The approach taken in attempting to realize these objectives included:

a) Define a functional specification language for defining prototypes. This language includes a data flow-like syntax and "programs" can be hierarchically decomposed to any desired level. The language was intended to be easy to use by non-programmers and the resulting programs to be directly executable by an Inter-

preter.

b) Organize the facility around the concept of reusability with the reusable components (both functions and data types) managed by an object manager. The reusable components were to be maintained in a library, and flexible tools were to be designed to allow users to browse the library, and build new specifications by extracting components from the library and inserting them into diagrams.

c) Develop a set of support tools to facilitate the construction and execution of prototypes. These tools were to include an Editor for building and modifying diagrams, an Interpreter for executing the diagrams, and a browser for locating desired components from the reusable library.

Some key concepts in the approach taken in implementing Proto include:

* a close coupling between the Proto human interface and the Proto database so that what the systems analyst sees on the display is what is in the database, and so that tools can share the objects and screen images with consistent interpretations;
* support for managing the layout of multiple data flow graphs including moving, shrinking, and scaling them - enabling the systems analyst to extract the view he/she wants of a particular multi-level hierarchical dataflow description; and
* support an object-oriented style of interaction, whereby each icon on the Proto display corresponds to a Proto database object whose class defines the type of editing/interpretation operations available for that object and thereby defines the content of the "pop-up" menu which guides user actions for that object.

3.3 Database Management System

The approach taken in the DBMS effort was to:

(1) identify database issues in software requirements development, characterizing requirements data and data access patterns (usage of requirements data),

(2) from the analysis in (1), prototype a DBMS that would serve as the Proto database, coupling the Proto tools, permitting a sharing of requirements data across Proto tools, and

(3) on the basis of experience gained from the Proto DBMS effort, specify the kind of DBMS needed to tightly couple requirements tools in the RET.

SECTION 4. RESULTS

4.1 The Requirements R&D Plan Effort

In this section we summarize the panel's recommendations and provide an overview of the long-term R&D program the panel defined to introduce new methods and tools into the RET and to aid in their evaluation. Appendix A provides more detail on the R&D program. Appendix A also lists the panel members and describes the requirements engineering process model the panel defined as part of its activities. Detailed results of the panel are described in [2].

4.1.1 Panel Recommendations

The Panel recommends that RADC pursue a research and development program for the Requirements Engineering Testbed (RET) consisting of two tracks:

(1) an Evolutionary Track for developing tools and methods such as rapid prototyping that in the near term give the best payoff in better requirements, and

(2) a Formal Language Track for exploring the higher risk/payoff implications of a formal requirements language. The risk in the Formal Language Track is that one must be able to express requirements formally. The payoff is in the formal activities that can be automated. Determining requirements satisfaction and generation of scenarios are examples.

The panel further recommended a testbed integration plan for 1990, that would lead to a uniform user interface to all tools and a common repository for all requirements, designs, and tool data, and which incorporates RADC's currently-contracted requirements tools. This recommendation is based on a short-term goal of a loose coupling of the tools.

4.1.2 Specific goals for 1990 and 1995

The R&D program consists of R&D thrusts in these areas: (1) prototyping, (2) requirements analysis, (3) tool integration and evaluation, and (4) a formal language for requirements and specifications. The objective of the Evolutionary Track is to provide better capabilities and more automation in the first two areas and to accomplish and support the third area. The focus of the Formal Language Track is the last area. Below, we summarize each area, identifying capabilities to be developed by 1990 and 1995, and the benefits.

Prototyping

In prototyping, the 1990 goal is to develop capabilities in: (1) prototyping system interfaces and functions, (2) developing scenarios to drive the prototypes in experiments, and (3) collecting and analyzing results. The 1995 goal is to extend capabilities for analyzing prototyping results.

Benefits: End users are generally able to analyze their mission concerns better in operational settings than by reading lengthy, textual requirements specifications. Thus prototyping will play an important role in discovering and understanding critical needs, and therefore in capturing these needs in the requirements. Prototyping will also provide a feasibility check, and when later supplemented with performance analysis, will serve as a basis for sensitivity analysis on requirements.

Requirements Analysis

The 1990-1995 goal for requirements analysis are to learn how to represent requirements more precisely, to strongly couple the analysis to requirements updates, and to identify relevant metrics. Another goal is to provide a methodology to guide the user in the use of these analysis capabilities.

Benefits: Requirements correctness and quality will improve. Air Force users will use the analysis tools to investigate their concerns in the requirements including the rationale and implications of decisions that were made.

Tool Integration and Evaluation

In the integration and evaluation of tools, the 1990 goal is to provide state-of-the-art database and human interface capabilities as a basis for an integrated testbed of tools and for monitoring tool performance. The 1995 goal is to integrate new prototyping, requirements analysis, and formal language capabilities into the instrumented RET to facilitate their evaluation.

Benefits: A common database implies tools will be able to share data. A common user interface implies user learning time will be reduced. Both imply the user will be able to move easily from one tool to another. Development efforts will be reduced - new tools can use the same data management facilities and utilize the same human interface mechanisms for user communication. An instrumented and integrated testbed is a basis for evaluating relative tool effectiveness.

Formal Language

In the formal language area, the 1990 goal is to provide: (1) a common formal language for requirements and specifications and (2) tools that automatically compare requirements and specifications statements. The 1995 goal is to provide: (1) the capability to generate and interpret scenarios, and (2) extensions to the language that facilitate incremental modifications and multiple levels of abstraction.

Benefits: Requirements statements will be precise and machine interpretable, leading to increased automation and support for prototyping, requirements analysis, and development activities.

4.2 The Prototyping Effort

The Proto system was implemented and delivered to RADC. As part of the evaluation, several demonstration prototypes were constructed including:

Advanced Sensor Exploitation (ASE)

This demonstration is of the Target Report Correlation component of the ASE problem.

Atmospheric Defense Initiative (ADI)

This demonstration was designed to work in conjunction with the RPS component of the RET. In particular, the Proto demo allows the user to control the movement of aircraft on the situation display provided by RPS. This demonstration was constructed, but has not yet been interfaced to RPS.

Library

This example manages the handling of books in a library. It handles check out and returns of books, adding and deleting books to the library, card catalog searches, etc. This example has been used for training analysts who plan to develop prototypes using Proto. It is documented in the Proto User's Manual [5].

The following subsections outline some of the key features of the Proto system. Appendix C gives a more pictorial view of the system and its features. The "screen dumps" in that appendix are taken from the ASE example.

4.2.1 Functional Specification Language

A functional specification language was implemented which:

* has graphical data flow-like syntax - a notation familiar to most systems analysts
* permits hierarchical decomposition - a means of employing abstraction in complex functional descriptions, and
* eliminates low-level design decisions - to improve productivity

The language has proven adequate for the functional prototype examples we have constructed. However, as discussed in Section 5, the language needs to be enhanced to be truly useful for prototyping complex real-time systems.

4.2.2 Reusability

The Proto system was built around the concept of reuse. In particular, the system:

* Includes a "Reuse library" which is maintained by an Object Manager. The library contains components that are used as building blocks for prototypes.
* Includes keywords as attributes of reusable components. Searches of the reusable library can be made based on these keywords.
* Includes facilities by which reusable components can be located based on defined relationships (e.g., all the functions that make use of a particular data type). This helps a user determine how a component is used in other contexts.
* Includes in the library both application-independent elements (e.g. parameterizable data management functions and forms interface functions), and application-specific elements (e.g., C3I functions). This is intended to improve productivity through reuse and to reduce the required level of computing skill required to produce a functional prototype.

The reusable library that has been constructed has proven quite adequate for the examples we have constructed with Proto. There is however still much work to do especially in the area of clarification and categorization of components, graphical browsing of the component library and editing components into existing graphs.

4.2.3 Support Tools

The three principal support tools provided by Proto are the "Diagram Editor, the Interpreter and the Browser.

Diagram Editor

A diagram Editor was developed through which users can construct executable functional specifications. The Editor understands the syntax of the diagrams and supports the user through a variety of consistency and completeness checks. Objects in the diagrams can have multiple alternative representations. These representations include:

dataflow - an object in a diagram can decompose into another diagram.

text - an object can be represented by a textual description (this aids in understanding a diagram, but is not directly executable).

code - an object can have associated source and object code. The object code is executable.

script - a high-level language that is directly interpretable.

Interpreter

A Proto Interpreter was developed that allows direct execution of Proto diagrams (graphs). In doing this, the Interpreter analyzes the graph to be executed and schedules component functions in the graph for execution. As the functions are executed, the Interpreter collects outputs from them and passes them along to downstream functions; and updates the schedule defining the order of execution. The Interpreter provides debugging aids (e.g., setting of breakpoints and single step operation) and capabilities for instrumenting (animating) the diagrams. During execution, the user can observe the sequence of execution of components of a diagram; additionally, through the instrumentation capabilities, the user can continuously view the values of selected data items.

Browser

A Browser was developed which allows a user to navigate through the database of objects. Objects and their attributes can be viewed and modified through this interface.

4.3 The Database Management System Effort

An analysis was performed to identify database issues that affect the requirements development process and based upon this analysis an approach to providing data management support for the RET was proposed. The proposal is to use an Object Manager (OM) to support all the RET tools. A discussion of the analysis and the resulting proposal are included in [8]. The executive summary for this report is contained in Appendix B.

The basic conclusion of this work is that requirements engineering information must be shared amongst multiple tools and the this can be done best by a model that treats the information as a collection of objects each of which contain explicit data structures and relationships to other objects. Furthermore, while this model can be supported by a relational or network database, an object-oriented database is preferred both because it will provide the most natural way to work with objects, and due to the complex relationships among requirements objects, it has potential for much better performance than other data management models.

The Proto system was implemented using an OM. One reason for using this approach was to help us evaluate the applicability of object-oriented technology to the requirements engineering problem. The results of the Proto experience reinforce the appraisal of the OM advantages. Several tools within the Proto system effectively shared data and object-oriented data displays to provide very smooth user transitions from tool to tool. It is clear that no other DBMS data model would have been as effective in conveniently storing the multiple degrees of composition and abstraction used in Proto.

SECTION 5. CONCLUSIONS

5.1 Conclusions of Panel

The conclusion of the Panel was that RADC pursue a Requirements Engineering Testbed (RET) program consisting of an Evolutionary Track and a Formal Language Track.

The "Evolutionary Track" proposes an evolutionary R&D effort to extend the current formalisms and tools. Initial efforts are toward the development of tools for prototyping interfaces and functionality, and in deriving performance estimates based on estimated or simulated work loads. Future efforts would develop tools and methods that aid in: (1) scenario development, analysis, and execution; (2) cost, risk, and performance analysis; (3) the acquisition, modeling, and usage of domain information; and (4) requirements analysis methodology.

The Formal Language Track proposes research effort be spent toward developing a single formal language for expression of goals, requirements, and solution architectures. With this language, users could automatically generate prototypes from a formal specification.

5.2 Conclusions from Proto Development

The initial implementation of Proto was a prototype intended to determine:

a) the effectiveness of a very high level graphical language for requirements development,

b) the software tools and human interface features required to support prototype development,

c) the role of component reusability in prototyping.

While Proto has not yet received extensive in-field use, so its prototyping role has not yet been fully evaluated, some conclusions from initial experience can be drawn in each of these three objective areas.

Effectiveness of VHLL for Requirements Expression

The choice of an effective system specification language is an area of continued research, and there are several difficult criteria to be met by such a language:

a) precise semantics to enable a direct prototype derivation;

b) a simple syntax, possibly graphical in part, so the proposed system can be described to prospective users and implementers, and possibly to improve the productivity of system specifiers; and

c) abstraction mechanisms to prevent the need for detailed program design decisions.

Our experience with Proto indicates that an executable graphical data flow-like language with hierarchical decomposition capabilities is a good starting point for a functional specification language. When defining top-level system specifications users find it very natural to talk in terms of functional components and interconnections (information flows) amongst the components. The ability to execute the specification to determine how well the prototype meets the functional requirements is highly desirable.

Proto has proven to be quite adequate for the test cases in which it has been used. The language provides a good communications medium between analysts and users of systems. Additionally, it properly focuses the analyst on abstract datatypes and functions early in the specification process, and discourages users from jumping into design level issues early in the process.

While the Proto language is very good for defining high-level system functionality, after decomposing a system down to the level where a specific algorithm is to be performed on a specific processor, Proto is not terribly helpful if the desired algorithm is not contained in the reusable library. This however is not a condemnation of Proto, rather it tells us that the capabilities of the reusable library need to be expanded substantially (e.g., it should contain user interface generators, mathematical programming languages, models for communications channels, etc.).

The Proto system has precise semantics and a very simple syntax. This has proven to have both advantages and disadvantages. On the plus side, users can learn to use the language quickly and do not have to worry about the execution order of the components in a diagram - namely, the scheduling is completely automatic. This feature helps accomplish the goal of not requiring the user to make detailed design decisions while implementing a functional prototype.

On the negative side, Proto uses a specific scheduling algorithm, and there are times when this algorithm does not execute complex diagrams in the order that the user might consider intuitive. If the user does not like the order in which the scheduler executes a diagram, the only way the user can affect the execution order is by changing the topology of the diagram - i.e., there are no controls provided to modify the default scheduling algorithm.

One of the key shortcomings of the Proto language is that it is currently single-threaded. To model complex real-time systems, the language should be modified to allow multiple processing paths to be active simultaneously. Providing a multi-threaded execution algorithm is also key to providing performance analysis capabilities

for the system.

Features that allow the user to directly control the execution order and multi-threaded execution capabilities are being planned.

Support Environment Including Human Interface

We believe that the overall approach taken in building Proto was a correct one. In particular, the object-oriented interface supported by an object manager works quite well. With this interface, the user points at the object of interest and sees a menu of options that are meaningful to the selected object in the current context (e.g., Editor versus Interpreter). Additionally, the approach of separating graphic displays from software tools that operate on the underlying information has worked quite well. With this approach the user sees a graph representing a prototype system on the screen. The user executes the desired tool to operate on the prototype (e.g., Editor), and the Editor "attaches" itself to the existing display. Edit operations change information in the objects and the object displays are automatically updated through methods that belong to the individual objects. The Editor itself does not "own" the display. When the user switches to another tool (e.g., the Interpreter), the display on the screen does not change.

The Editor tool has proven adequate to build Proto diagrams. It understands the syntax of a diagram and supports the user by not allowing meaningless diagrams to be created. The Editor provides a number of advanced editing features to simplify the task of the user.

While the Editor is adequate, a number of improvements should be made to make it more user-friendly. Some of the more desirable features not yet implemented include a flexible cut and paste facility, the ability to split and merge multi-level graphs, and user definable icons. Note, the cut and paste required is more complicated than a simple "graphical" cut and paste since the objects being manipulated have an underlying semantics that must be accounted for during the operation.

The Proto Interpreter has significant limitations (e.g., the single threaded operation and fixed scheduling algorithm), but within these limitations it functions very nicely. The ability to watch the functional prototype run (e.g., the bubble(s) currently executing are highlighted), the ability to set graphical breakpoints, and the ability to instrument diagrams (i.e., display the values of selected data items along the arcs over which the data flows) have proven to be of great assistance in both debugging and evaluating the functionality of prototypes.

The existing implementation of the Proto Browser provides very basic capabilities for viewing and and modifying objects in the reusable library. It is coupled with the other tools in the sense that the user can point at an object and ask to browse it; however, the Browser creates its own viewing window rather than operating directly on the diagram being displayed. While the current capability is adequate for the prototype, a

more integrated graphical browser should be developed.

Role of Reuse

The Proto system depends heavily on its library of reusable objects - in fact all data-types, all executable diagrams, and all the components from which diagrams are composed are elements from the reusable library.

The existing library contains both domain independent objects (e.g., data manipulation functions and user interface definition tools) and domain dependent functions (e.g., objects that were developed specifically for the individual examples that were constructed).

Experience has shown that a user can build a prototype quickly if the reusable library contains most of the functions required for that particular example. To date the test examples used have been from different domains, and hence most of the reusability has been through the use of domain independent components. The domain dependent components have been provided through a combination of modules written in Proto's high level Script language and modules written in C.

Our conclusion from using Proto is that the reusable library approach is completely viable; however, a great deal of work remains to be done. This work should address reusability from different directions:

Domain specific library

> A specific domain should be selected (e.g., C3I) and a broad range of reusable components for that domain should be created. After such a library is in place it will be possible to evaluate the effectiveness of reuse by determining how many of the underlying components have to be developed from scratch when a new system from within the selected domain is to be prototyped.

Tools to create/modify components

> A suite of tools should be developed that simplifies the task of creating new reusable components and tailoring existing components to solve new but similar problems.

Enhanced browsing

> As the library of reusable components grows, it will be important to have better tools for locating and viewing components that are candidates to use for a new problem.

Coupling to external tools

> Various external tools can be viewed as reusable components that the user can "insert" into prototypes. For example, if the proper interface software is developed, RPS can be viewed as a tool for creating reusable components intended for modeling user interfaces, situation displays, etc. Other external tools such as graphical mathematical programming languages and graphical user interface generation tools should also be integrated together with Proto to support the development of reusable components.

5.3 Database Manager for RET

Based on the Database Management study and the results of the Proto development effort, our conclusion is that an Object Manager should be used to manage all requirements engineering information in the RET. This model will provide a natural way to define the artifacts for representing the information required by the requirements engineering process and will provide the optimal way for managing the complex relationships among these artifacts. Other models (e.g., the relational, hierarchical or network models) could be used but they would suffer both in terms of performance and ease of use as compared to the object model.

BIBLIOGRAPHY

[1] Rzepka, W., Ohno, Y., "Requirement Engineering Environments: Software Tools for Modeling User Needs", IEEE Computer, April 1985.

[2] International Software Systems, Inc., "RADC System/Software Requirements Engineering Testbed Research and Development Program", RADC-TR-88-75, Griffiss AFB,, NY, June 1988.

[3] Stephens, M., Whitehead, K., "The Analyst - An Expert Systems Approach to Requirements Analysis", Proceedings 8th Int'l Conference on Software Engineering, London UK, August 1985.

[4] Konrad, M., Welch, T., "VHLL System Prototyping Tool User Manual", Air Force Contract F30602-85-C-0129, Griffiss AFB, NY, June 1987.

[5] Rzepka, W., Daley, P., "A Prototyping Tool to Assist in Requirements Engineering", Proceedings 19th Hawaii Int'l Conference on System Sciences, Honolulu, HI, January 1986.

[6] Konrad, M., Hartman, D., "Functional Description for Proto", Air Force Contract F30602-85-C-0129, Griffiss AFB, NY, January 1988.

[7] Welch, T., Konrad, M., "Database Issues in Software Requirements Development", IEEE Database Engineering, March 1987.

[8] Welch, T., "A Data Management Facility for Requirements Engineering Testbed (RET)", Air Force Contract F30602-85-C-0129, Griffiss AFB, NY, December 1988.

[9] Konrad, M., Welch, T., "VHLL System Prototyping Tool - User Manual", Air Force Contract F30602-85-C-0129, Griffiss AFB, NY, September 1988.

[10] Hartman, D., Clendening, G., Leon, J., "Proto System/Subsystem Specification", Air Force Contract F30602-85-C-0129, Griffiss AFB, NY, August 1988.

[11] Hartman, D., Clendening, G., Leon, J., "Proto Program Specification", Air Force Contract F30602-85-C-0129, Griffiss AFB, NY, September 1988.

APPENDIX A. THE REQUIREMENTS R&D PLAN EFFORT

A.1 Panel Members

Members of the panel that defined the long-term RET R&D program were:

Robert Balzer - Information Sci. Inst., Marina del Rey, CA
* Michael Konrad - International Software Systems, Austin, TX
C.V. Ramamoorthy - University of California at Berkeley, CA
Winston Royce - Lockheed Missiles & Space, Austin, TX
* William Rzepka - Rome Air Development Center, Rome, NY
Steve Sherman - Lockheed Missiles & Space, Austin, TX
Leon Stucki - Future Tech., Auburn, WA
* Terry Welch - International Software Systems, Austin, TX
Raymond Yeh - International Software Systems, Austin, TX

---

* Rzepka was panel sponsor. Welch was panel chairperson. Konrad was panel report editor.

Pei Hsia of the University of Texas at Arlington, Texas contributed to the panel's efforts.

A.2 The Requirements Engineering Process Model

In an effort to come to a common understanding of the requirements engineering process, especially terminology, and to partially capture that understanding, the panel created a model of the process. This section presents that model.

Report [2] contains much more discussion and further details on the process model.

The model is portrayed in figure A-1. The figure shows information depicted as boxes; and activities depicted as circles, ovals, and rounded-corner boxes.

The model indicates the dependencies and sequencing between activities, but is not meant to favor a particular methodology.

Information Types

The model identifies three major types of information: goals, requirements, and solution architectures.

Goals are expressions of objectives and needs, generally mission-related, and not necessarily feasible or consistent with each other. Mission users are the primary source.

Requirements are a consistent subset of the goals which can be feasibly realized within the available resources (especially time, money, and expertise).

A Solution architecture is a model of the target system as a composition of parts that satisfy the requirements. The more common term is specification, but the panel preferred solution architecture, as specification has been used to mean different things.

A Walk-through

We take a top-down walk through the requirements engineering process model, identifying the objects and activities depicted in the figure. For the walk-through, we assume a requirements engineer is required to produce a set of requirements for a system called the "target system".

The target system must support the different roles of its users and administrators. Thus there will be different expectations, or "viewpoints" of what the system should do, of how well it should be done, and within what cost. In the model these undocumented expectations of target system functions, performance, and cost are represented by Wish Lists. There is one wish list per role or viewpoint.

The requirements engineer is limited in the time he can expend in the creation of target system requirements. Thus he needs to prioritize his objectives. These limitations and objectives should both be documented. In the model they are represented by Engineering Context Descriptions.

Through interviews with target system user/administrators and through references to documentation of similar, existing systems and their environments, the requirements engineer collects and organizes information on the operational context of the target system. The resulting information forms a Domain Model of the environment of the target system, providing the terminology and context through which wishes can then be expressed, forming Goals. Goals represent the initial attempt at documenting a system's desired attainments. For each viewpoint, there will be one set of goals, and they should be consistent and complete with that viewpoint.

Goals are often inconsistent across viewpoints or clearly infeasible. Such difficulties must be resolved by the requirements engineer through further user interviews. The revised goals are then merged into a preliminary set of Requirements for the target system.

Through his interviews with users, the requirements engineer identifies and documents Scenarios that illustrate typical target system behavior and/or desired responses to stressful input. Scenario construction and analysis may aid stating the nonfunctional requirements, in particular, performance and reliability.

During the creation of goals and requirements, their consistency and completeness is checked. This is called Static Analysis in the process

model. To determine requirements coverage, the requirements engineer might perform a walk-through, analyzing the dataflows and/or stimuli/responses through the various viewpoints. This is called Dynamic Analysis in the process model.

At this point, the requirements engineer might construct a Solution Architecture to gain better insight into: target system interfaces, functions, performance and reliability, and implied development cost and risk.

The requirements engineer creates a solution architecture by specifying how the target system is composed of parts (e.g. objects, functions) and how those parts use resources (e.g. people, software, hardware). To aid specification of resources, the requirements engineer can make reference to existing resource models.

From the solution architecture, the requirements engineer can specify a prototype. He executes the prototype against canned or user-controlled scenarios, eliciting user comments on what should be changed. All of these activities are covered by the Rapid Prototyping bubble in the process model.

Also, the requirements engineer can do Analysis directly on the solution architecture. By combining both rapid prototyping and analysis activities and iterating, the requirements engineer can do sensitivity analyses.

As a result of the insights gained through analysis and rapid prototyping, the requirements engineer determines what revisions should be made and makes them. This activity is called Requirements Evaluation & Reformulation in the process model.

There may be several iterations of prototype, analyze, evaluate and reformulate. The resulting requirements and solution architecture is called the Final Requirements and Partial Solution Architecture in the process model.

A.3 The RET R&D Program

Objectives

Below we summarize, the RET R&D program objectives:

* The R&D program should provide tools and methods that work together in the ways illustrated by the 1990 and 1995 scenarios (The 1990 and 1995 scenarios are not described in this report but appear in [2].)

* The R&D program should provide tools and methods to support process model activities.

* The R&D program should help fulfill these RADC goals for the RET: (1) The RET should support evaluation of the effectiveness of tools and methods. (2) The RET should make a full range of requirements engineering capabilities accessible to Air Force mission users and acquisition engineers. (3) The RET should host the currently-contracted tools, (Analyst, Proto, and Rapid Prototyping System) and by 1990, they should be integrated.

* Long-range architecture for the RET - The R&D program should realize an RET architecture featuring: (1) a direct manipulation-style user interface to all objects, (2) a database serving as the common repository for all requirements related information, and (3) a formal language for expression of goals, requirements, and solution architectures.

  In the long term, all RET tools and methods should be structured to fit this architecture.

References are made to these objectives in the sections that follow. Section A.3.1 summarizes the panel's strategy for obtaining an integrated RET. Section A.3.2 discusses an R&D program consisting of two tracks: (1) an Evolutionary Track for developing tools and methods that in the near term provide the best payoff in better requirements; and (2) a Formal Language Track for exploring the higher risk/payoff implications of a formal requirements language.

A.3.1 Near-term Integration of RET

To address RADC's objective of integrating the currently-contracted tools, the panel recommends that integration be achieved by having the tools work offa common database and be accessed through a common user interface. This level of integration means that: (1) tools can share data, and (2) the RET user is given uniform access to tools and their data and is free to invoke tool functions in an order natural to his/her application.

This approach will produce an early version of the long-range RET architecture.

An integrated RET will also help in the evaluation to tools; for example, by providing the basis for a broader range of control experiments.

To significantly reduce the amount of effort required to achieve integration, the panel recommends a near-term strategy of standards and cooperation between the RADC tool contractors. The integration strategy is further discussed in [2].

To provide RET users some of the benefits of integration in the very near term, the panel recommends a loose coupling of the currently-contracted tools. The loose coupling plan is also discussed in [2].

A.3.2 Requirements Engineering Testbed (RET) Research and Development Program

Two-Track Program

To meet the objectives stated at the beginning of section A.3, the panel identified two themes on which RET R&D program efforts should focus: (1) providing near-term support for these activities: prototyping, requirements analysis, and evaluation of tools; and (2) a formal treatment of requirements. The R&D program consists of two tracks to deal with these two themes. Figure A.3.2-1 depicts the R&D program road map the panel defined to realize these themes. Below, we expand on these themes and their associated activities and then discuss the figure.

Theme of the Evolutionary Track

Both the 1990 scenario and the process model characterization demonstrate the importance of prototyping and the role of scenarios in driving prototypes. Prototyping gives a mission user "visibility" into specifications of system and software requirements by helping the mission user determine whether his/her needs are being addressed. Thus the panel recommends that the "creation of prototypes and scenarios, and analysis of results" activity be an early focus of the RET R&D program.

Some of the A.3 objectives imply the need for tools and methods that address the non-solution-architecture phases of requirements engineering; specifically, goals and requirements synthesis and analysis. Such tools and methods would help mission users state their needs and decisions at the mission level. Both scenarios demonstrate the need for such capabilities. Such a need must be satisfied in the near term. Thus, the panel recommends that the "analysis on requirements" activity be another early focus of the RET R&D program.

Evaluating the effectiveness of RET tools and methods requires the capability to track their use and collect results. An integrated RET would help control independent parameters (e.g. style of presentation, format of input data), a prerequisite for parallel experiments. Thus the panel recommends that the "measurement of tools in an integrated RET" activity also be an early focus of the RET R&D program.

To successfully provide near-term support for the three activities above requires a low risk and early payoff strategy. In the long term, the resulting RET capabilities would be enhanced/refined. The panel organized an "Evolutionary Track" of R&D efforts to do this. The Evolutionary track would provide the capabilities illustrated in the

1990 scenario and support most process model activities. The Evolutionary Track is described in detail in [2].

The Theme of the Formal Language Track

The panel recognized early on that representing requirements in a formal language was a high-payoff approach, but such an approach would fail to provide near-term solutions to the R&D program objectives. Nevertheless, such an approach would address these objectives not being addressed in the other track: (1) help automate requirements traceability and assessment of requirements coverage, and (2) provide a language for representing requirements, solution architecture. The panel thus defined a "Formal Language Track" whose focus would be to provide such a formal language.

The Formal Language Track would also provide the capabilities illustrated in the 1995 scenario and support most process model activities.

The Figure

Figure A.3.2-1 depicts 1990 and 1995 goals of the two themes: creation of prototypes and scenarios and analysis of results ("Prototyping"), analysis on requirements ("Analysis"), measurement of tools in an integrated RET ("Tool Evaluation"), and formal language. Their dependencies with each other and with the currently-contracted tools are indicated by the arcs.

Analyst tool capabilities are the basis for the 1990 Analysis capabilities for structuring requirements and the domain model. Domain models provide necessary information for building scenarios and simulations, thus the vertical dependency with 1990 Prototyping goals. 1990 Prototyping goals are also dependent on the prototyping and scenario generation capabilities provided respectively by the Proto and Rapid Prototyping System tools.

1995 Prototyping goals extend 1990 Prototyping capabilities by providing capabilities for sensitivity analysis on requirements. 1995 Analysis goals extend 1990 Analysis capabilities by providing capabilities for dynamic analysis and quality critiquing. Various interactions are possible between 1995 analysis tools for these two activities, hence the double-headed vertical arrow.

The 1990 Tool Evaluation standards will be influenced by the standards adopted by developers of the currently-contracted tools. These 1990 standards will in turn guide all subsequent RET tool development. The 1995 Tool Evaluation goal is to integrate these new Analysis, Prototyping, and Formal Language capabilities into the instrumented RET to facilitate their evaluation.

The 1990 Formal Language goals are independent of 1990 Analysis and Prototyping efforts and the currently-contracted tools. The 1995 Formal

Language goals include incorporating abstraction mechanisms into the language and interpreting classes of scenarios. These capabilities must also be integrated into the RET.

From the perspective of milestones, the 1995 RET will be a mature experimental facility, hosting matured analysis, prototyping, formal language, and evaluation capabilities. The 1990 RET will be a prototype of the 1995 RET, still integrated and instrumented for evaluation, but featuring only a few mature tools, in particular, the currently-contracted tools.

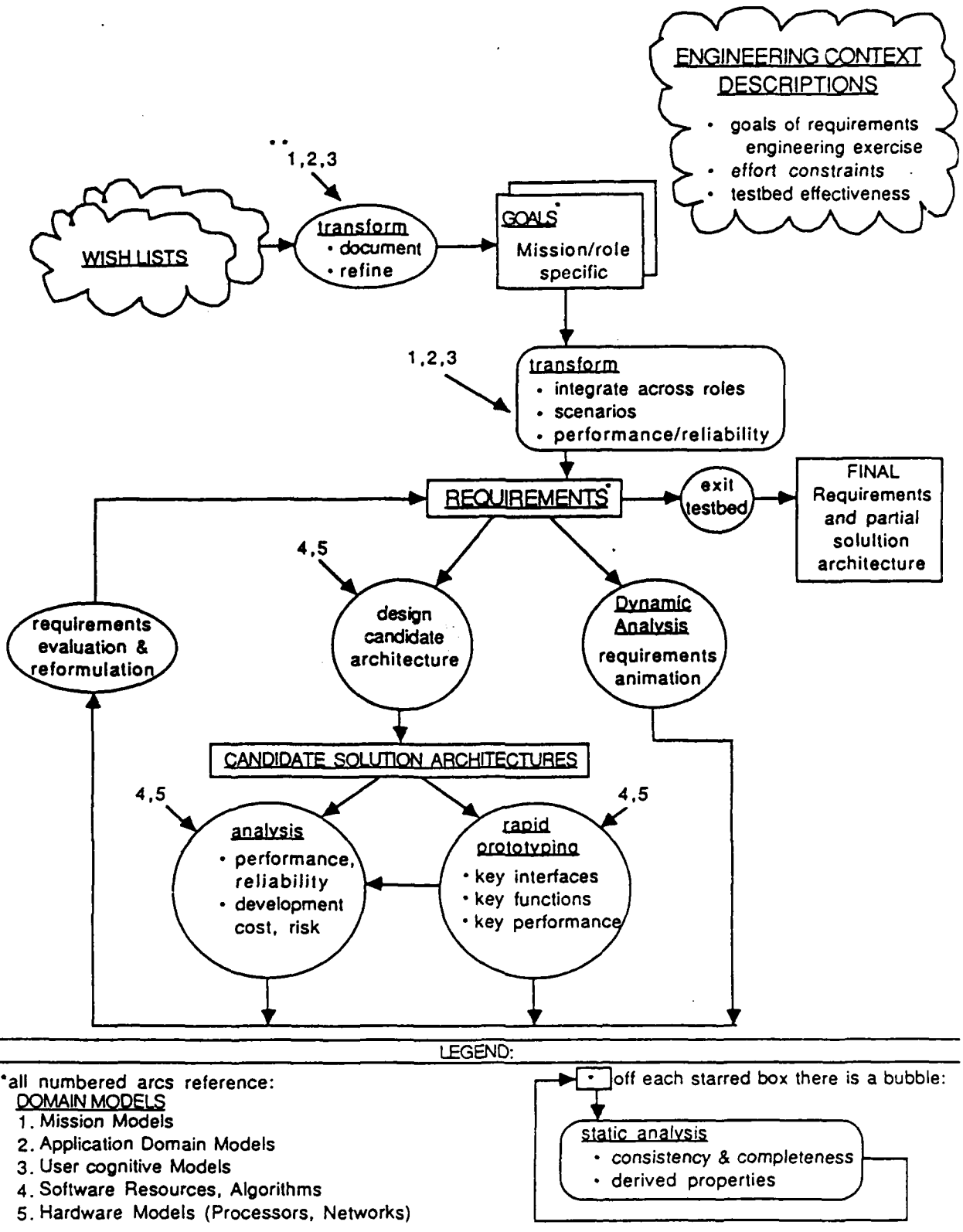


Figure A-1 Requirements Engineering Process Model.

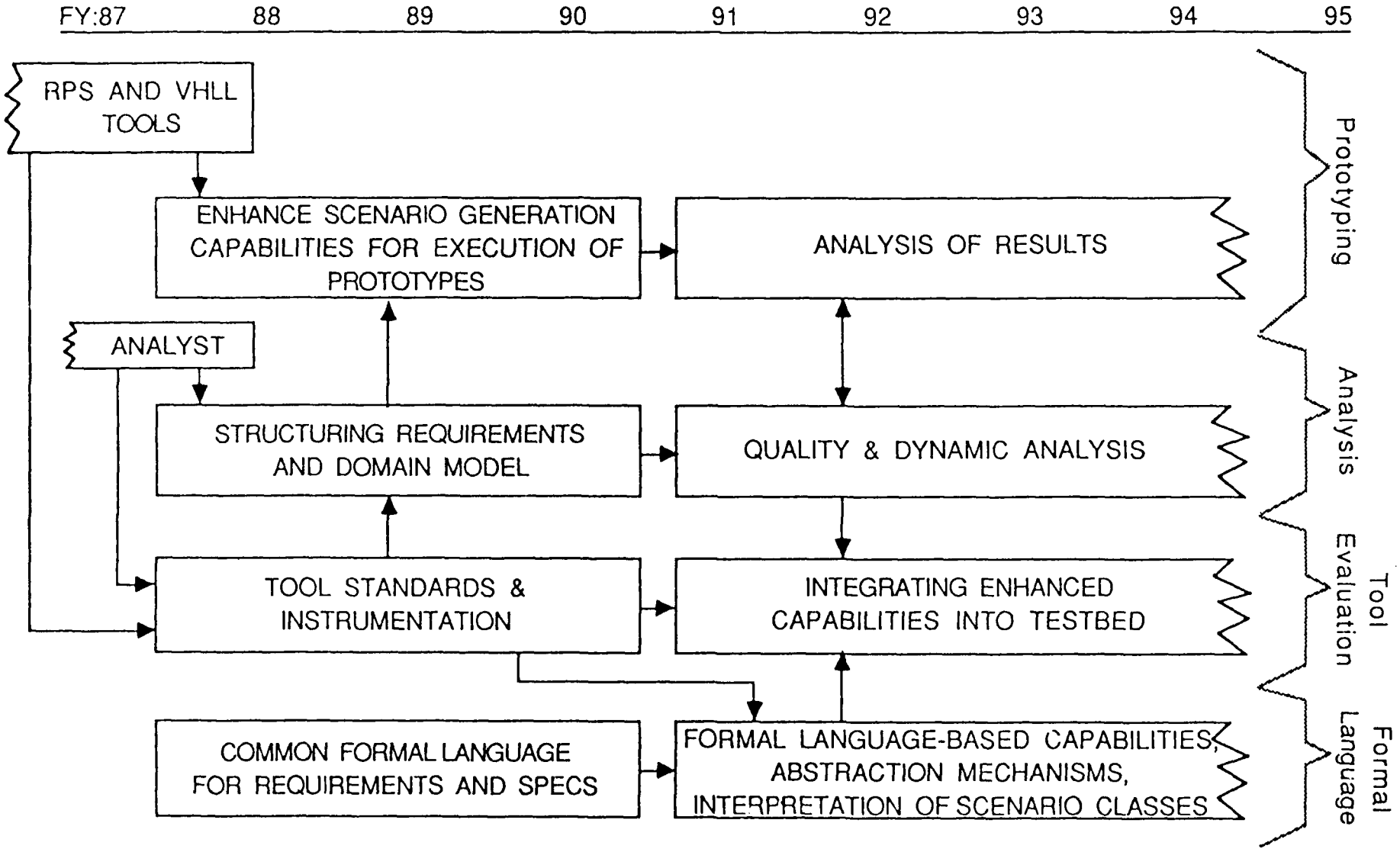


Figure A.3.2-1 RET R&D ROADMAP

APPENDIX B: Executive Summary from Database Study

The following is the executive summary taken from reference [9], "A Data Management Facility for RET".

The DBMS for RET is called upon to provide a path for sharing data amongst three existing tools and potential future software engineering tools. A line of argument is developed which shows that specification descriptions should be shared between tools in a form where the relationships between operators and data structures are explicit, as opposed to the relatively implicit relationships contained in textual descriptions (e.g., conventional programming languages). This leads to a database representation of entities (objects) and their relationships to each other. Objects can be composed into dynamic structures (e.g., a hierarchy), so a conventional relational DBMS proves to be awkward and inefficient for general object storage. The object model effectively supports the extraction of tool specific data as "views" of the stored data, so that various tools can share data without having to conform to the same data structure.

An Object Manager (OM) differs from a relational DBMS principally in that it provides an identifier for each object instance and permits a wider variety of attribute types. It shares some of the implementation problems of any DBMS in choice of logical and physical tool interfaces.

The proposed object manager will provide object definition and manipulation capabilities along with object storage. The Object Manager has the potential for better performance than a relational DBMS because the related objects are more likely to be clustered and the relationships between the objects are used more effectively.

We recommend that the OM be implemented in a separate process, in the Unix sense of process, for protection of the data base. A tool or other application program would call procedures to insert and retrieve objects, and those procedures would then communicate with corresponding procedures in the OM process. Data from the OM process is loaded into an "in-memory" object model associated with an individual tool. The in-memory representation is organized for more efficient access to object information and for more rapid movement among related objects. This strategy reflects tradeoff examinations of ease of tool development and modification, execution speed, and flexibility of data sharing.

APPENDIX C: Proto Capabilities and Concepts

Rapid prototyping is pursued as a means to develop and validate functional specifications prior to extensive code development in a large software system. To achieve a functional prototype execution with moderate development effort three capabilities are desirable: 1) a functionality specification language, which minimizes the design decisions which must be provided by the prototype developer when describing prototype functionality; 2) a library of reusable software modules to expedite system specification, and a database system to aid in module retrieval and analysis; and 3) a set of interactive tools, including a system interpreter with debugging features, which aid in the construction and execution of prototyping experiments.

PROTO is a rapid prototyping system with these capabilities, providing a graphical prototyping language and its support tools. It employs a strategy of component reuse which incorporates object-oriented modules. It provides interactive tools which help users cope with complex system design information by means of graphical presentations.

In PROTO, one creates a functional prototype, which is used to validate the specification of system responses, to determine if the functionality is usable in the target context. This style of prototyping can also be used to verify design decisions, such as algorithm selection, interfaces, etc. This is distinct from behavioral prototyping, which checks the human factors aspects of man-machine interfaces, and performance prototyping which provides response times for a design that implements the target specifications. Functional prototyping is used to stabilize software requirements via demonstration of system capabilities to potential end-users.

The prototyping language uses a dataflow style of presentation to express precedence relationships between operations. It supports a hierarchical and object-oriented structuring. It supports abstract datatype concepts mixed with conventional data typing, but shields the user from data space management concerns.

The prototyping support environment is centered around an object-oriented data management system which is closely coupled to a workstation display. This structure is effective for storing and viewing hierarchical compositions of program graphs, data type definitions, and other design data. The object management system provides better access performance and simplicity of query expression than conventional data management systems do for complex design data.

We illustrate PROTO capabilities via a sequence of screen images in the following pages.

This work was supported by a Rome Air Development Center contract on rapid prototyping (No. F30602-85-C-0129).

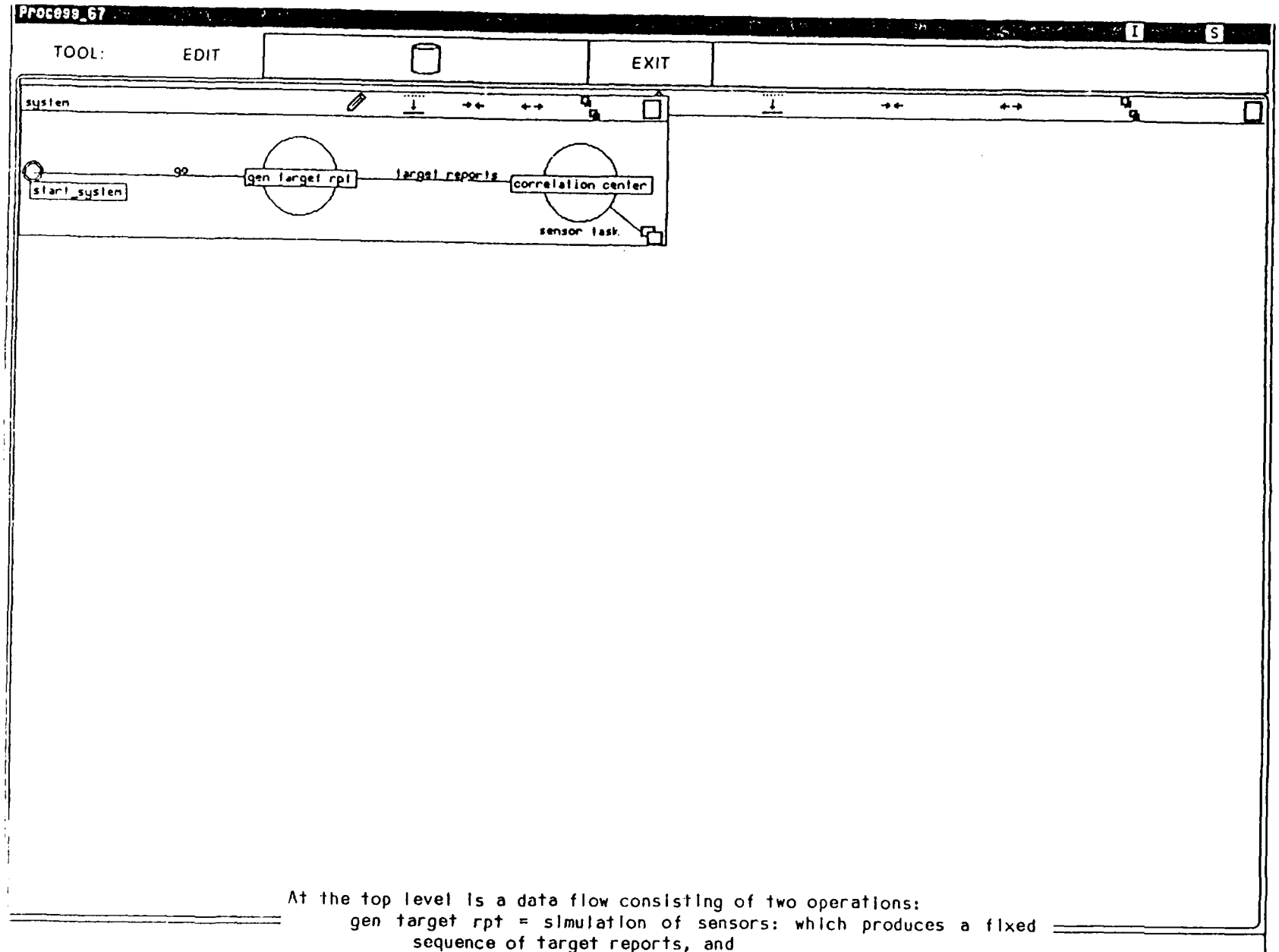
EXAMPLE PROTOTYPE

The following five screen images illustrate a particular example, Target Report Correlation. This is described here as a vehicle to demonstrate some capabilities of the Proto language and tools.

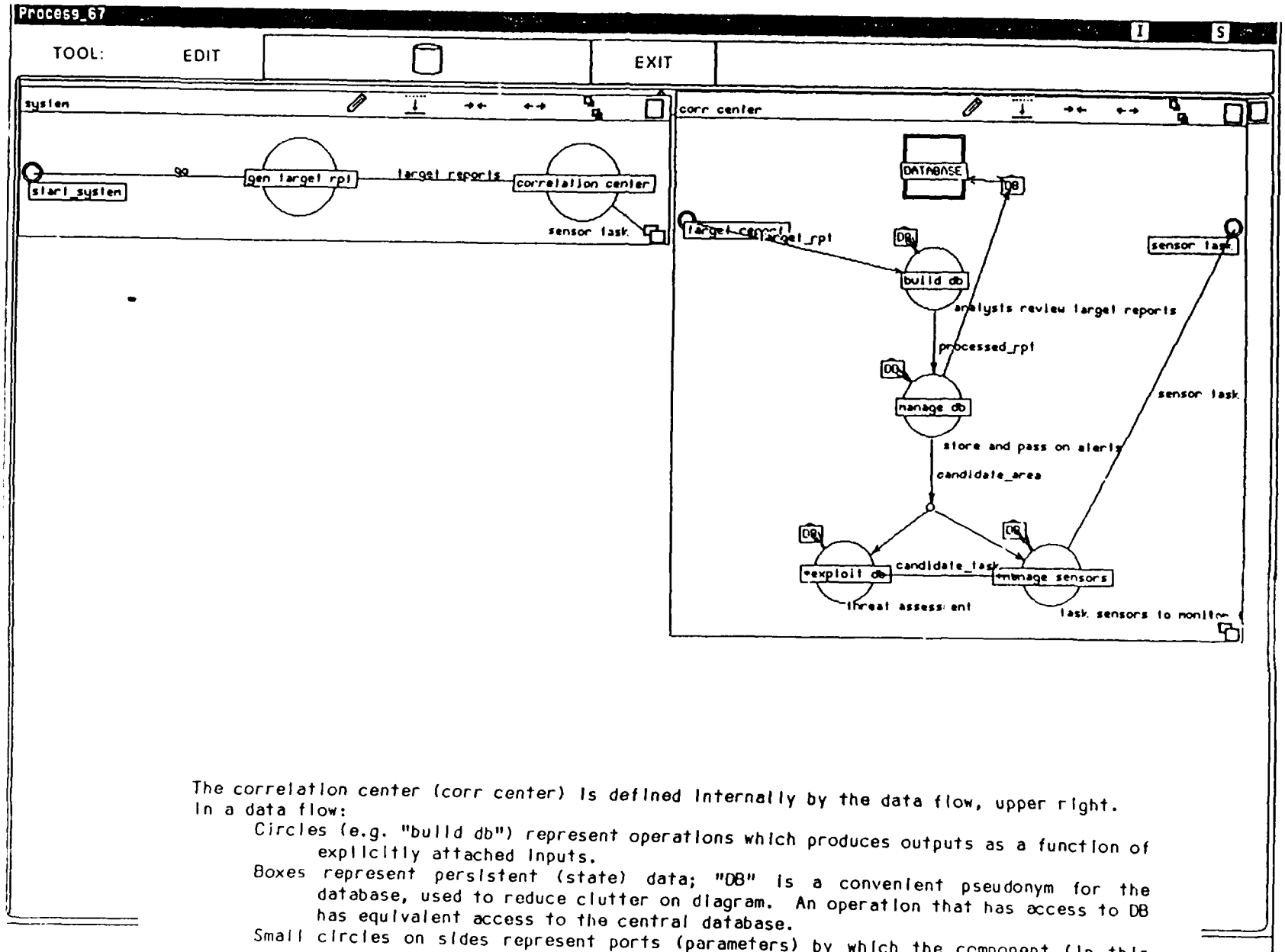
The example shows a series of target reports received from a sensor. For example, it might be satellite observations of tanks. Each target report specifies the position, direction, speed, and size characteristics of a target formation, and specifies the time the report was made. In what follows, each reported target formation is called a "group", and the group which is the subject of the current target report is called "the target".

The software system is to correlate successive reports to determine which reports determine a new position of a previously-reported group or when a new group has entered the view area. This correlation is interactive, with a human analyst making final correlation decisions based on information presented on an Analyst Workstation.

The objective of the prototyping effort would be to determine what information is needed by the analyst, and what operations he/she needs for effective decision making. The system also includes an automated correlation algorithm to aid the analyst, and prototyping would serve to determine the benefit of this help.



At the top level is a data flow consisting of two operations:
gen target rpt = simulation of sensors: which produces a fixed sequence of target reports, and
correlation center = software system being prototyped

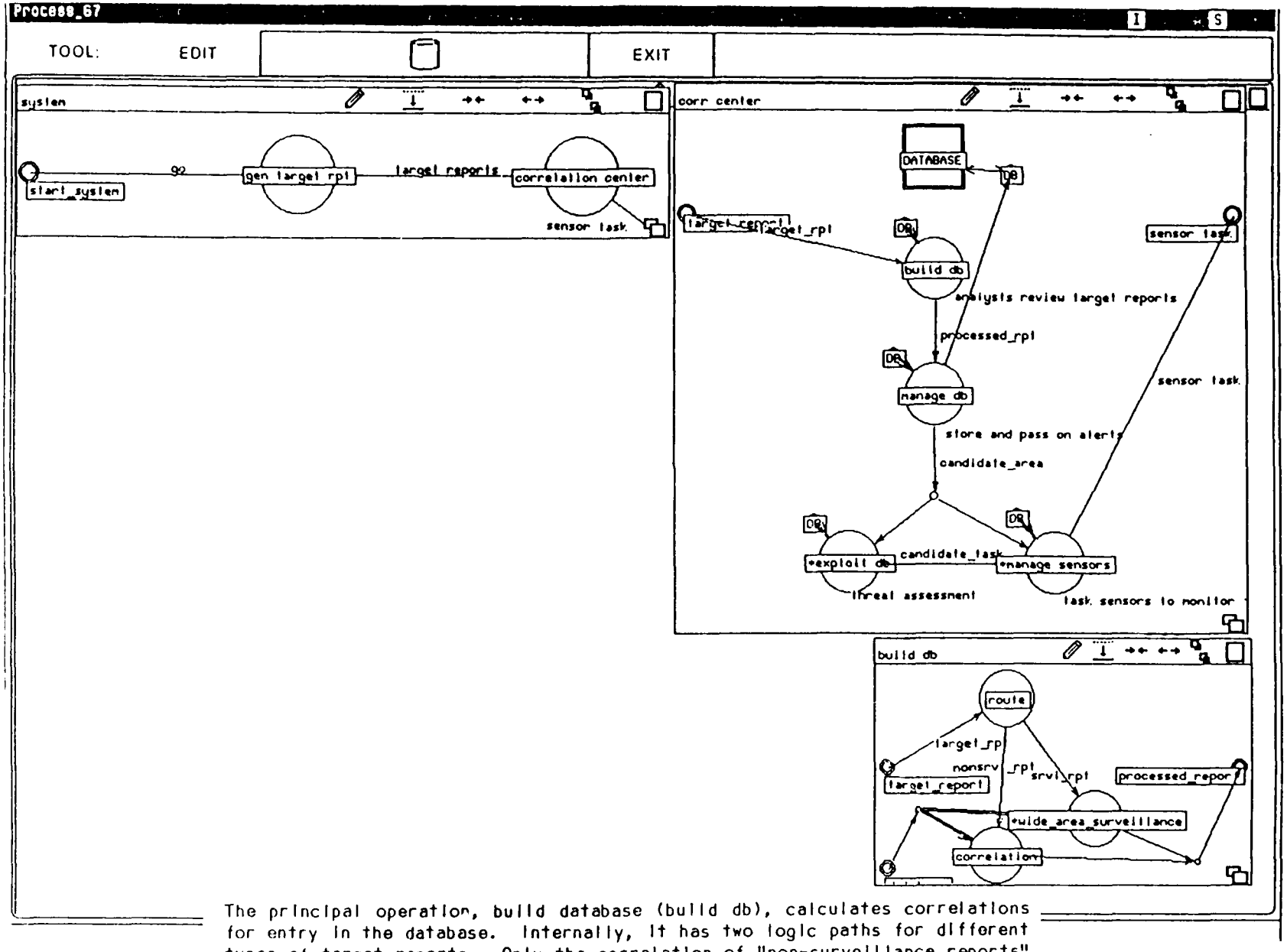


The correlation center (corr center) is defined internally by the data flow, upper right. In a data flow:

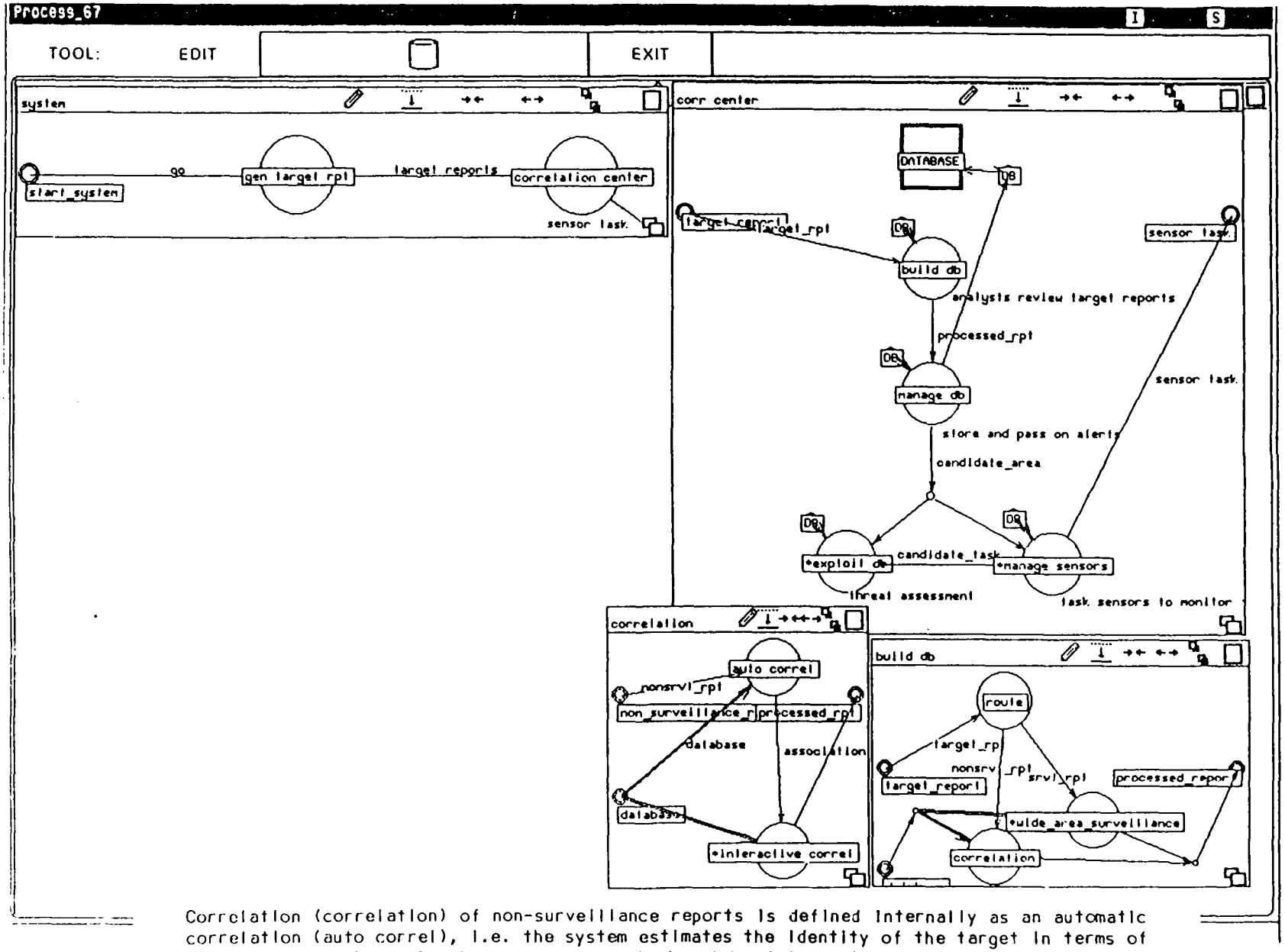
Circles (e.g. "build db") represent operations which produces outputs as a function of explicitly attached inputs.

Boxes represent persistent (state) data; "DB" is a convenient pseudonym for the database, used to reduce clutter on diagram. An operation that has access to DB has equivalent access to the central database.

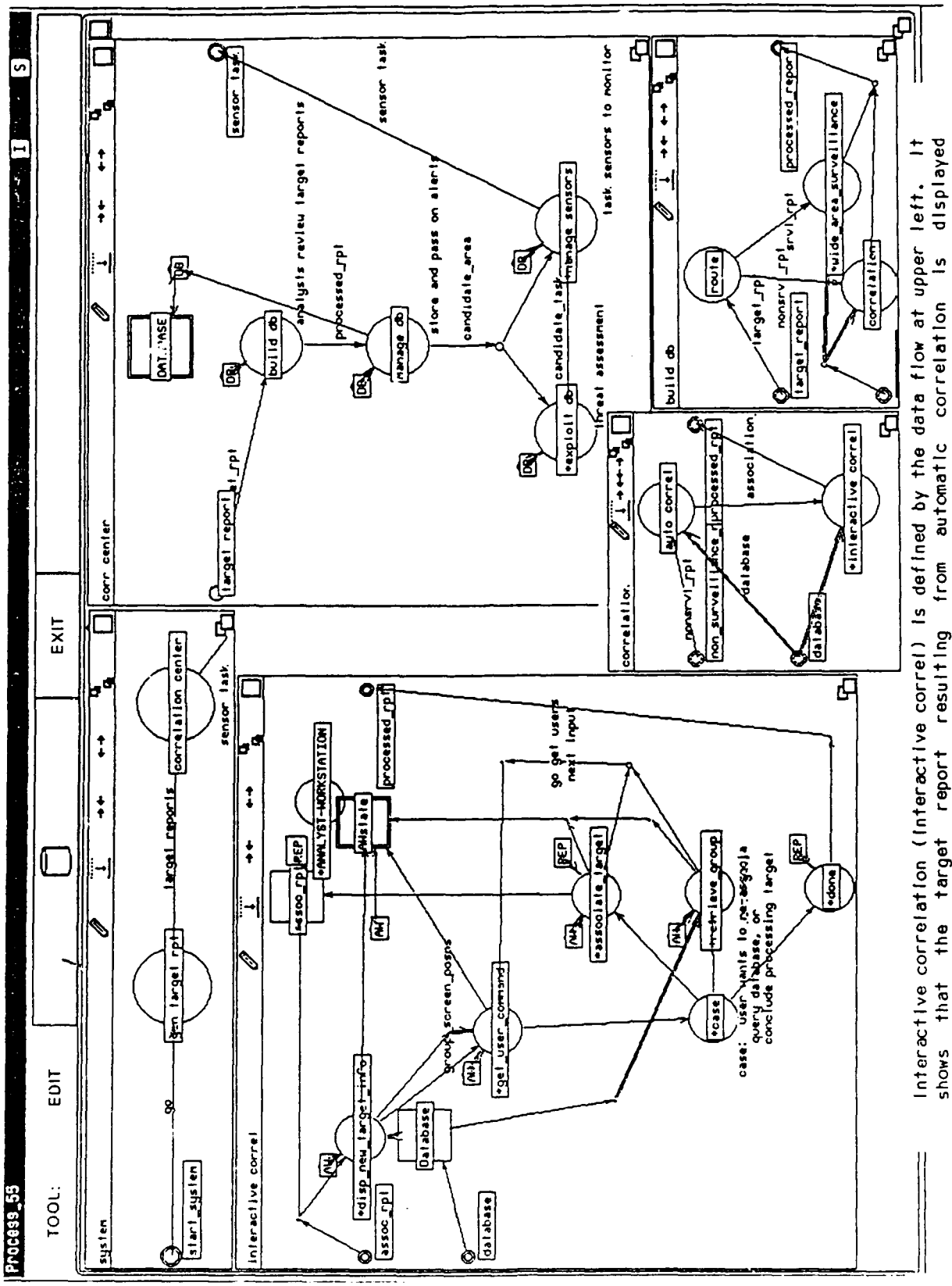
Small circles on sides represent ports (parameters) by which the component (in this case, correlation center (corr center)) connects to other components externally. Ports are datatyped.



The principal operation, build database (build db), calculates correlations for entry in the database. Internally, it has two logic paths for different types of target reports. Only the correlation of "non-surveillance reports" (nonsrv_rpt) is explored here.



Correlation (correlation) of non-surveillance reports is defined internally as an automatic correlation (auto correl), i.e. the system estimates the identity of the target in terms of the locations of previously seen groups, followed by interactive correlation (interactive correl), where an Analyst reviews and possibly modifies the systems's estimate.



Interactive correlation (Interactive correl) is defined by the data flow at upper left. It shows that the target report resulting from automatic correlation is displayed (disp_new_target_info). The analyst is given the opportunity to alter the automatically-correlated target estimate. Thus, analyst input is queried (get_user_command), and processed via a workstation represented here as object "ANstate".

PROTO SPECIFICATION LANGUAGE

The previous screen images illustrate the use of the PROTO language for functional specification. The key features of this language are reviewed here.

A Dataflow Diagram Language. The syntax is a refined dataflow diagram syntax, because as a visual form, dataflow diagrams appear to provide ease of understanding for non-programmers, and if used with care, provide a precise description of system functionality. They also serve as suitable inputs to the design process.

Activation Semantics. Proto has a adopted a message-passing model for dataflow diagrams, whereby each operation sends its outputs as messages which activate subsequent operations. The message-passing paradigm shields the system specifier from implementation issues of memory space management and operation sequencing. This simplifies the specification of a system relative to doing it in a high-level language such as Pascal. The result is that the prototype may not execute of highest speed, but prototype execution performance is assumed not to be a primary objective so long as it is not unacceptably slow.

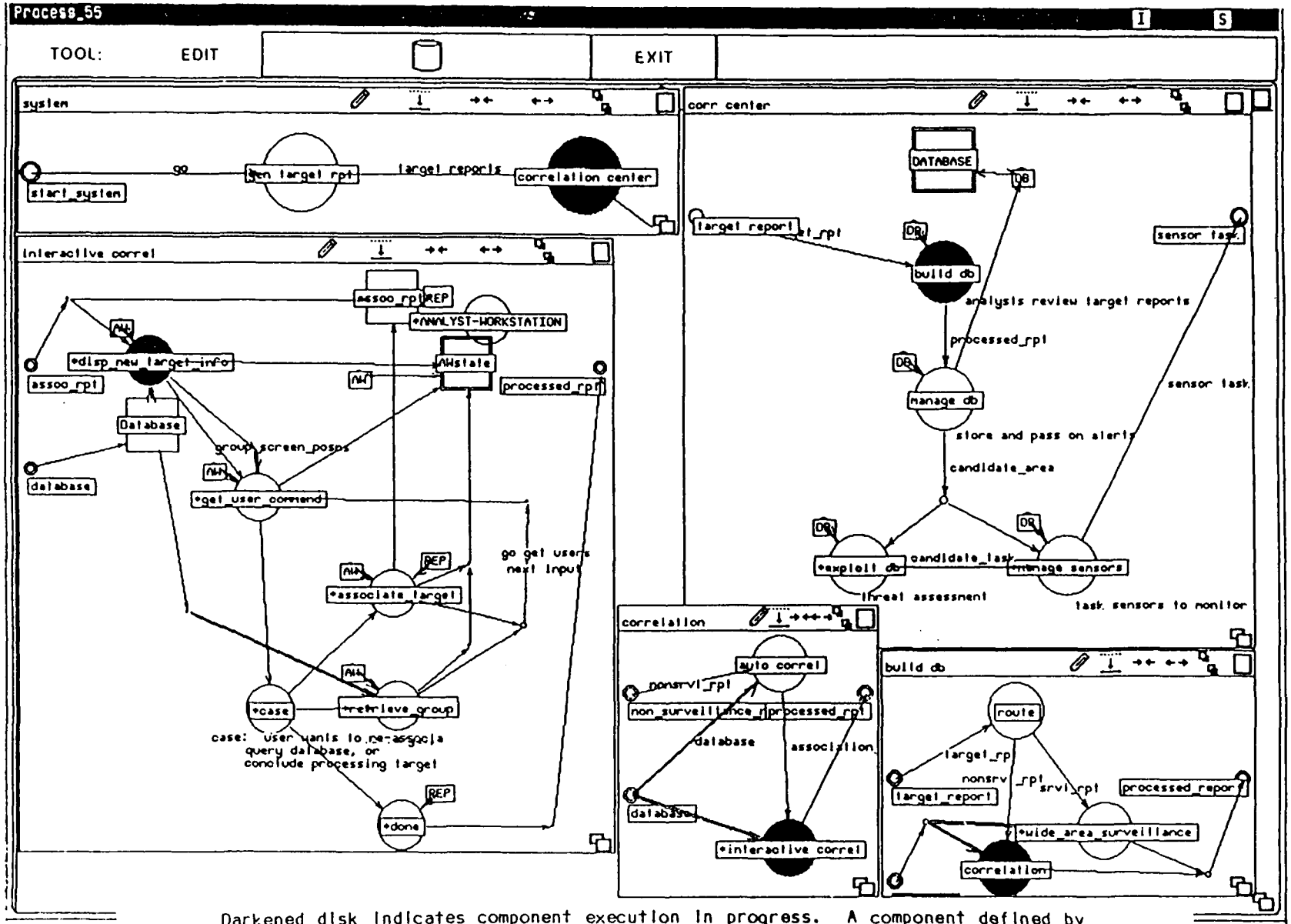
Proto specifications are essentially interconnections of components whose functionality can be defined three ways: 1) as Proto dataflow diagrams, 2) selected from a library of reusable components, or 3) defined by HLL source code, including a specialized language unique to Proto called "script".

SYSTEM ENACTMENT

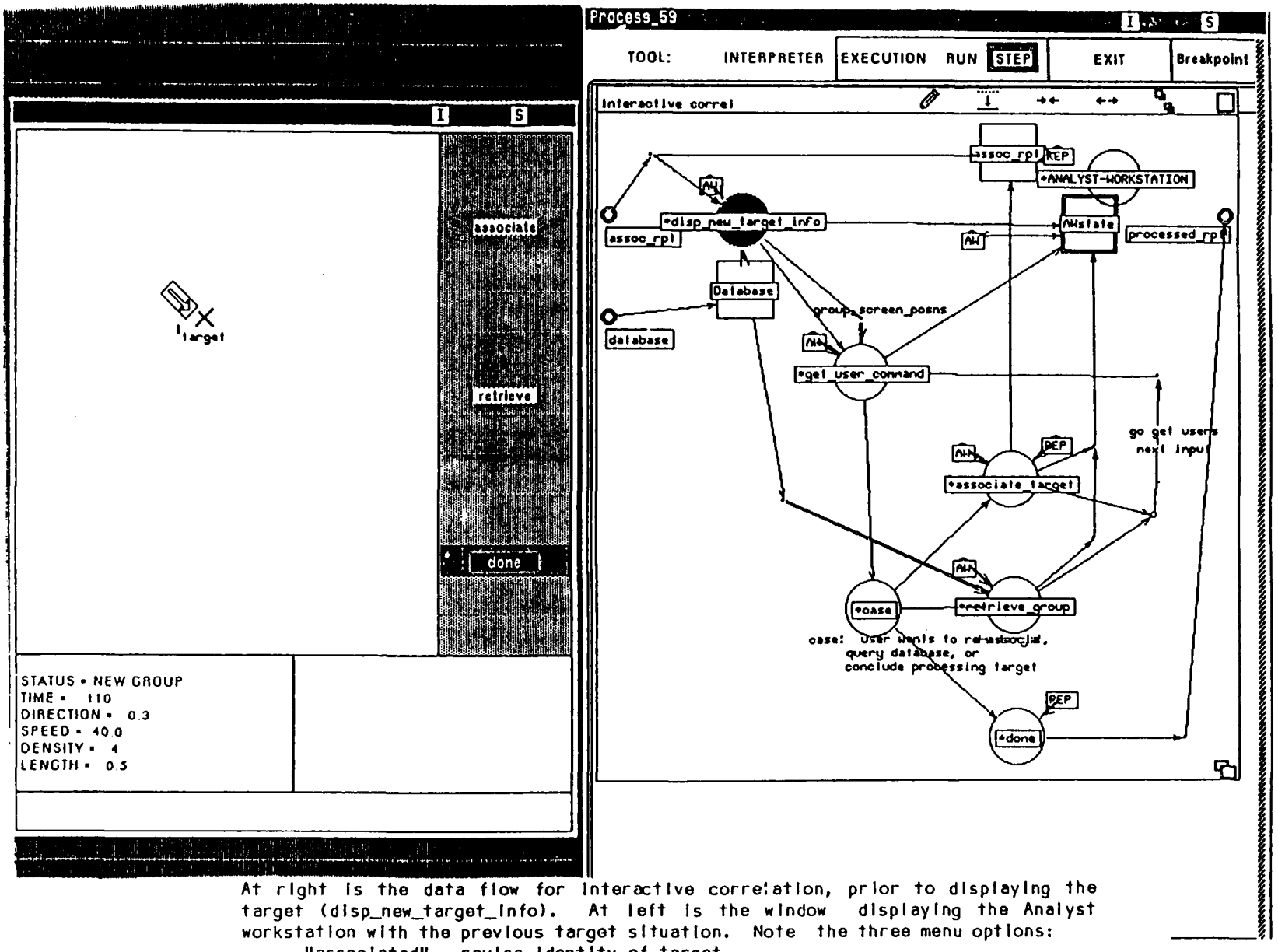
System enactment is achieved by direct interpretative execution of the dataflow diagrams. This enactment produces the specified system outputs for each set of selected inputs. Thus, multiple "input scenarios" can be applied and the specification reviewed in terms of the results produced. This enables one to exercise a prototype to see how it reacts to particular stressful inputs and to explore implications of "what if" questions.

Interpretation in PROTO produces a graphical animation of the processing and is useful as a debugging aid. Standard debugging support is also provided, such as breakpoints, single-stepping, etc. Interpretation can resume immediately after the re-editing of a specification, allowing timely exploration of user suggestions.

The next six screens show snapshots of an interpretation.

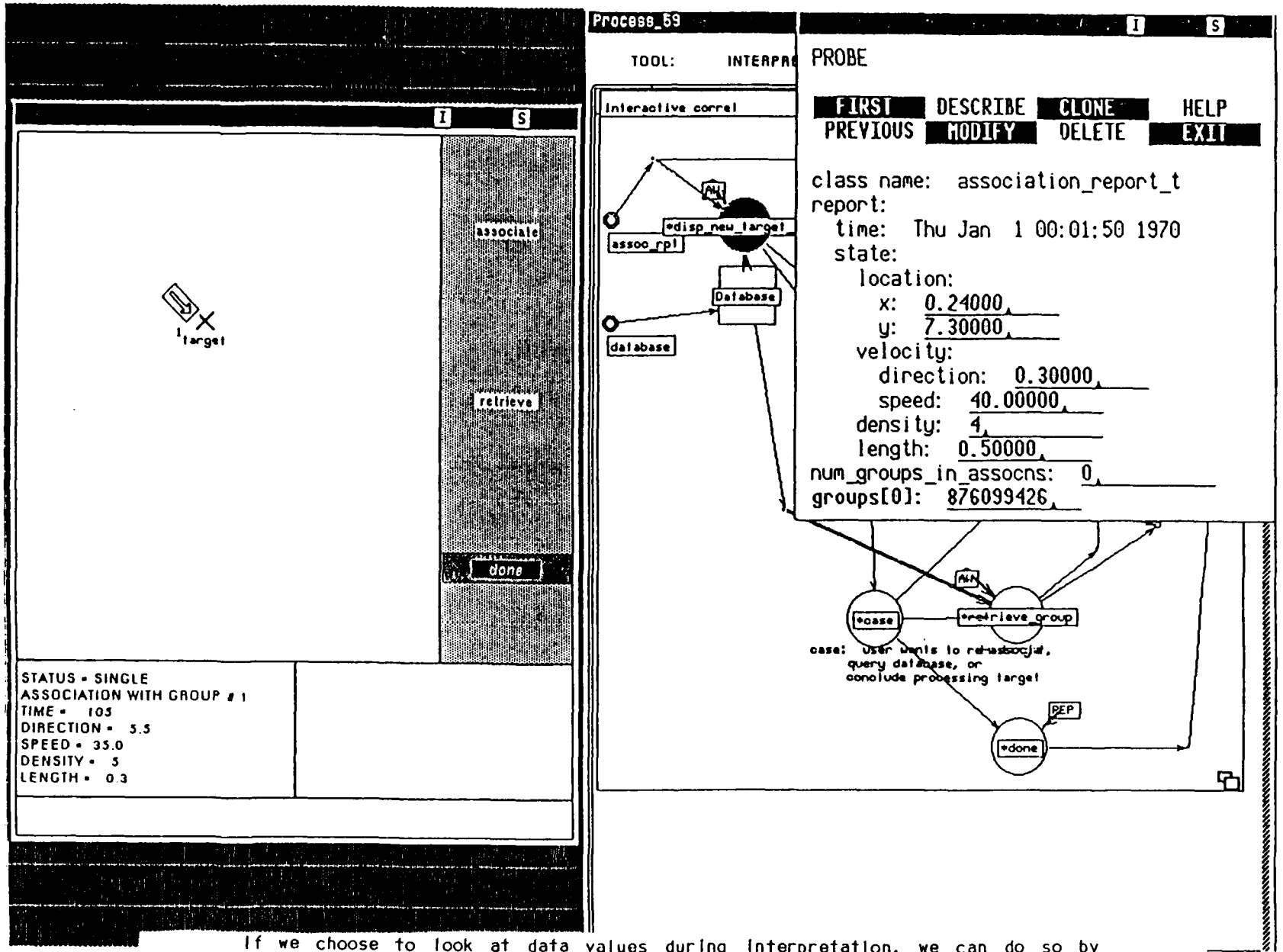


Darkened disk indicates component execution in progress. A component defined by data flow is executed by interpreting the data flow. As one part of executing "correlation center", we execute "build db", which requires execution of "correlation", etc. This shows a snapshot in time of which component operations are in active interpretation.

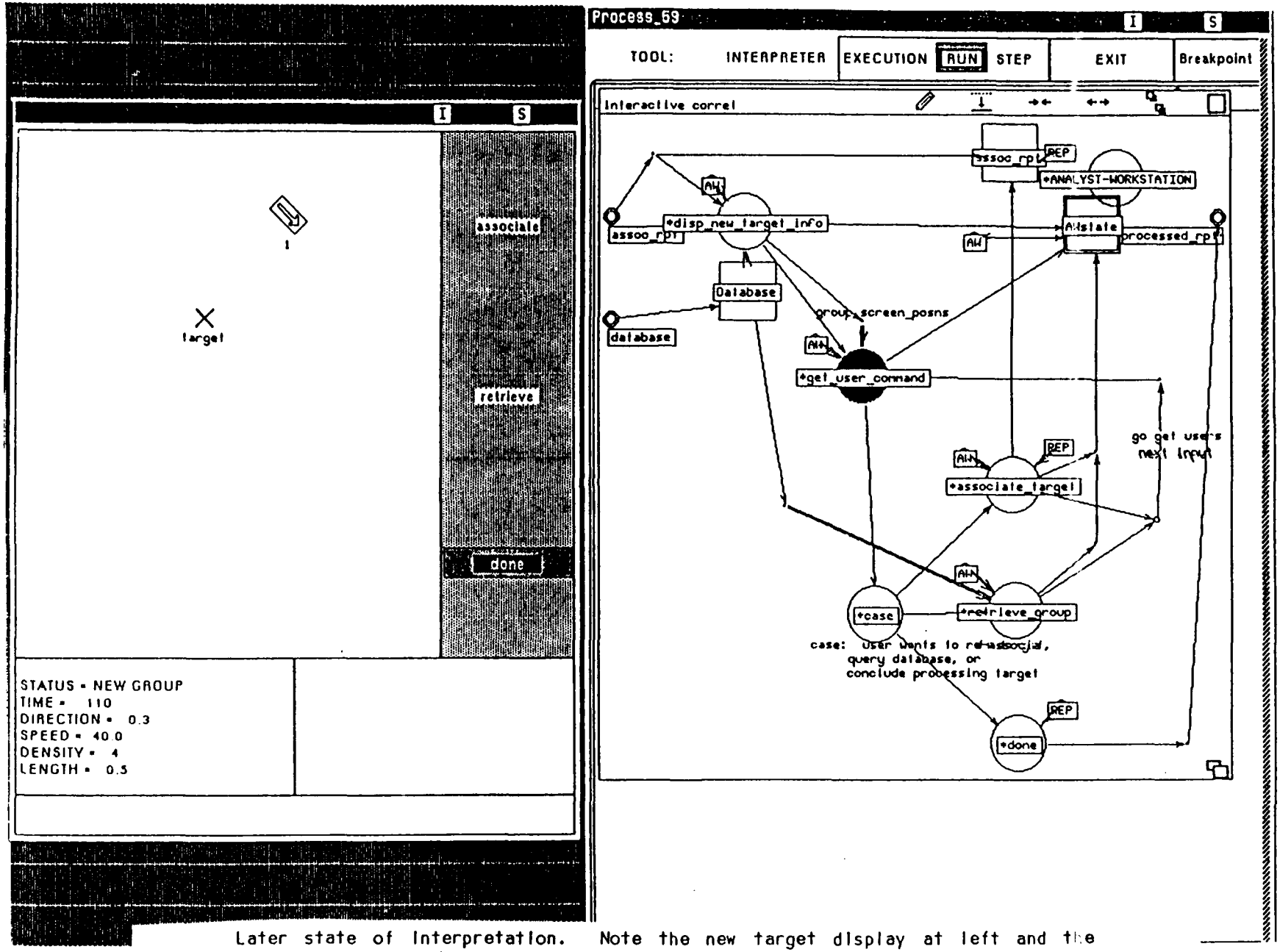


At right is the data flow for interactive correlation, prior to displaying the target (disp_new_target_info). At left is the window displaying the Analyst workstation with the previous target situation. Note the three menu options:

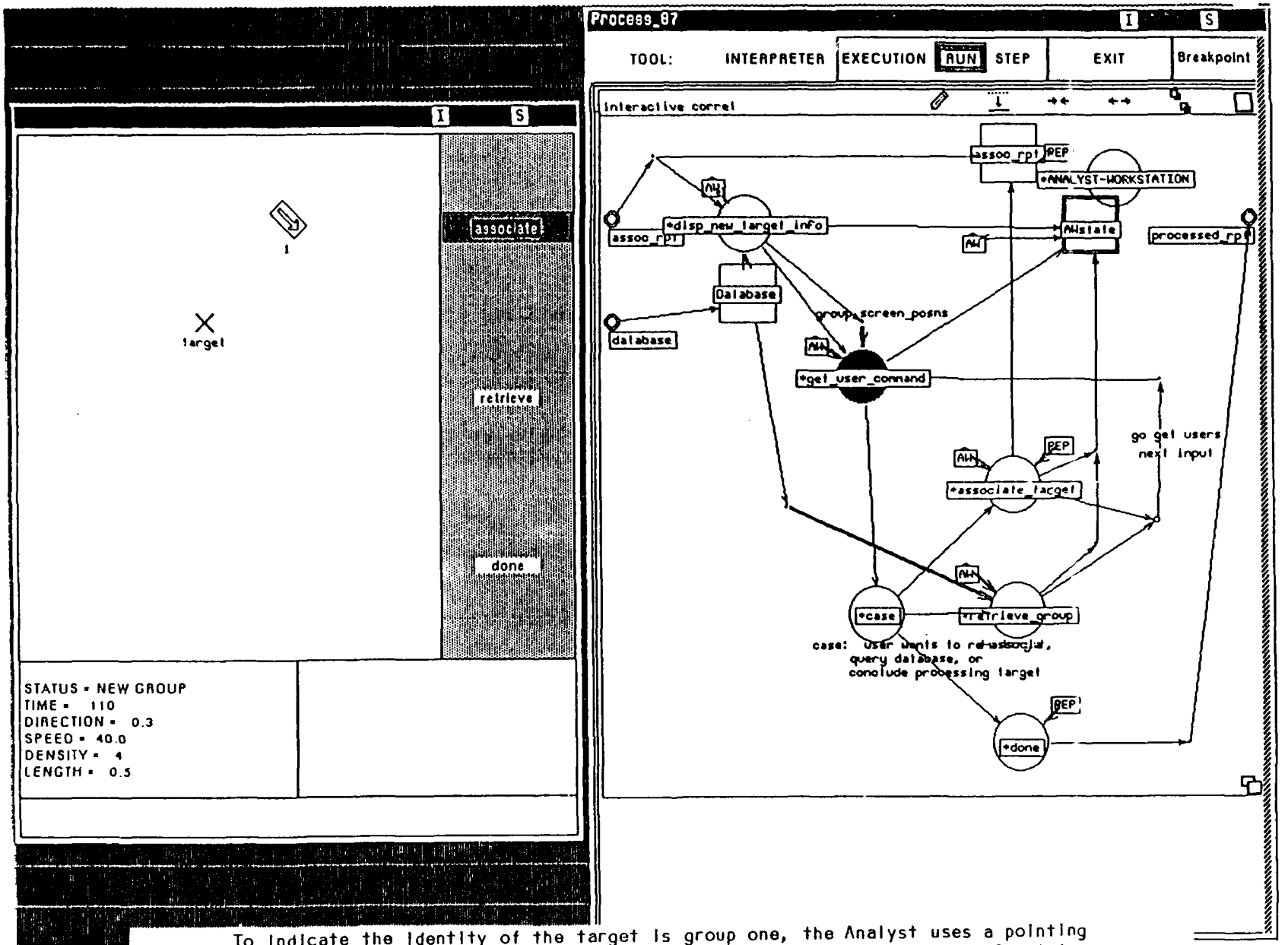
"associated" - revise identity of target,
"retrieve" - request information on known group (numbered arrow),
"done" - target correctly identified, get the next target report.



If we choose to look at data values during interpretation, we can do so by selecting the appropriate arc. A display is brought up indicating the data. The data can also be modified at this point.



Later state of interpretation. Note the new target display at left and the darkened component (get_user_command) at right, waiting for Analyst input.



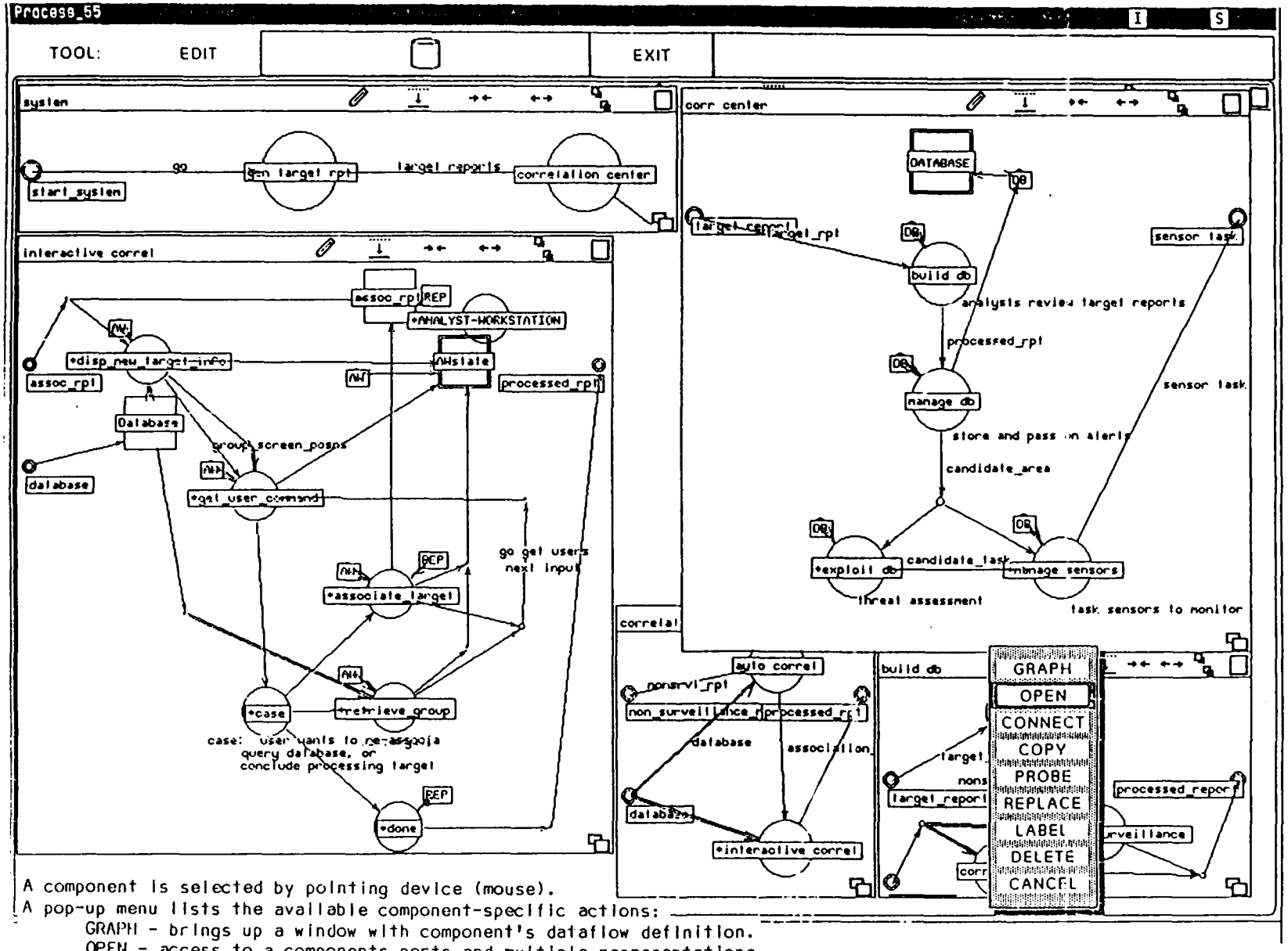
To indicate the identity of the target is group one, the Analyst uses a pointing device to select the arrow labeled "1" and the menu option "associate". Receiving this input, execution of the darkened component (get_user_command) completes.

EDITING

The creation and modification of specifications must be convenient and fast for prototyping to be rapid. Part of the challenge for an editor is meaningful presentation of potentially very complex system specifications in a form that is easily edited.

Presentation of specifications requires extracting views from a multi-level hierarchical specification, in which each level can be described in several dimensions: diagrams, textual specifications, interface specifications, code, etc. The interactive graphical facilities of a workstation provide effective means to let the user navigate and form a view of a specification. PROTO exploits workstation capabilities by a flexible windowing system which allows the user to manage the layout of information on the screen.

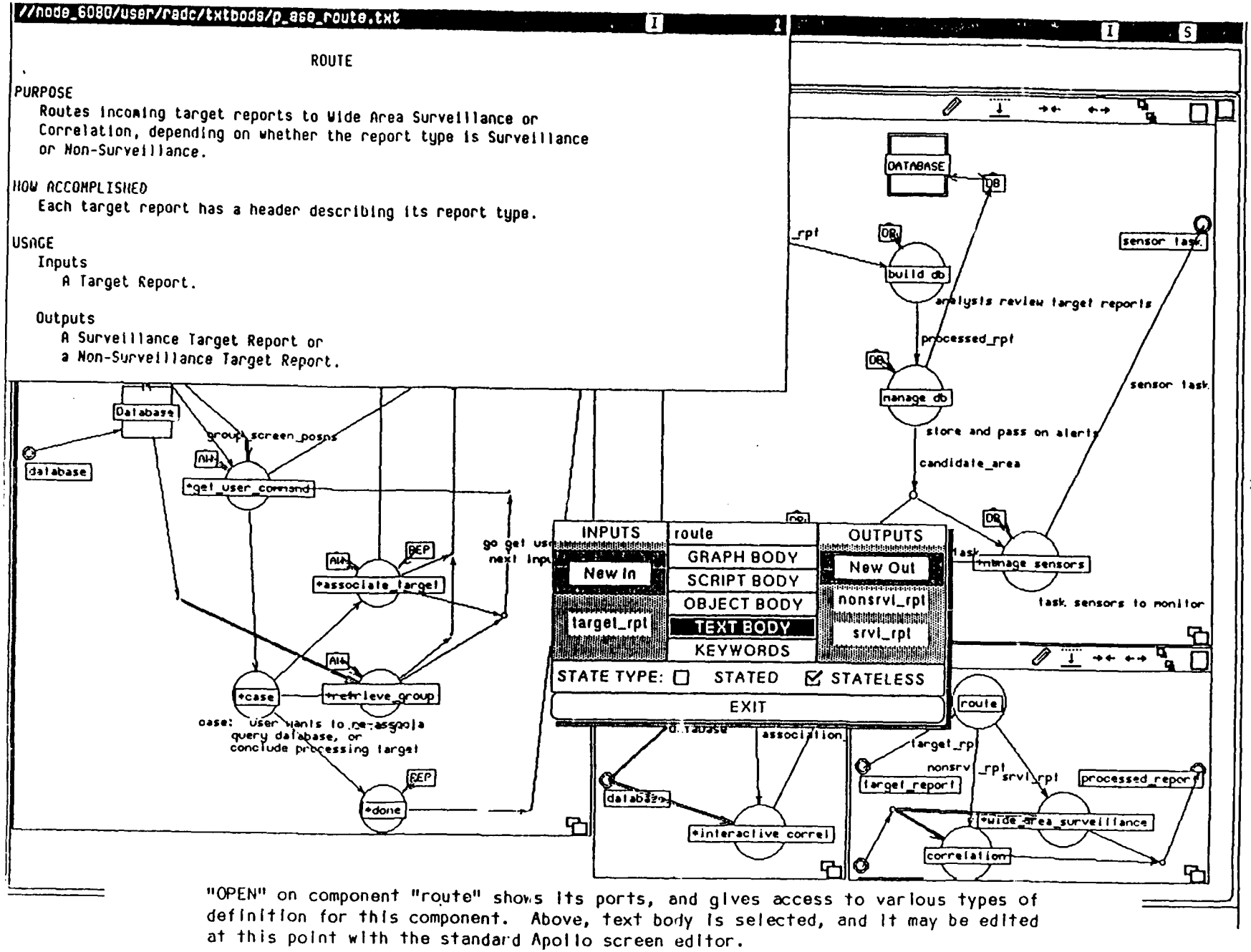
Editing of Proto graphs is facilitated by an "object-oriented" style of interaction, whereby each icon on the workstation screen corresponds to a database object whose class defines the type of editing available for that object, and thereby defines the content of the "pop-up" menu which guides user actions for that object. Modifications are made directly to the database object, for immediate access by other tools.



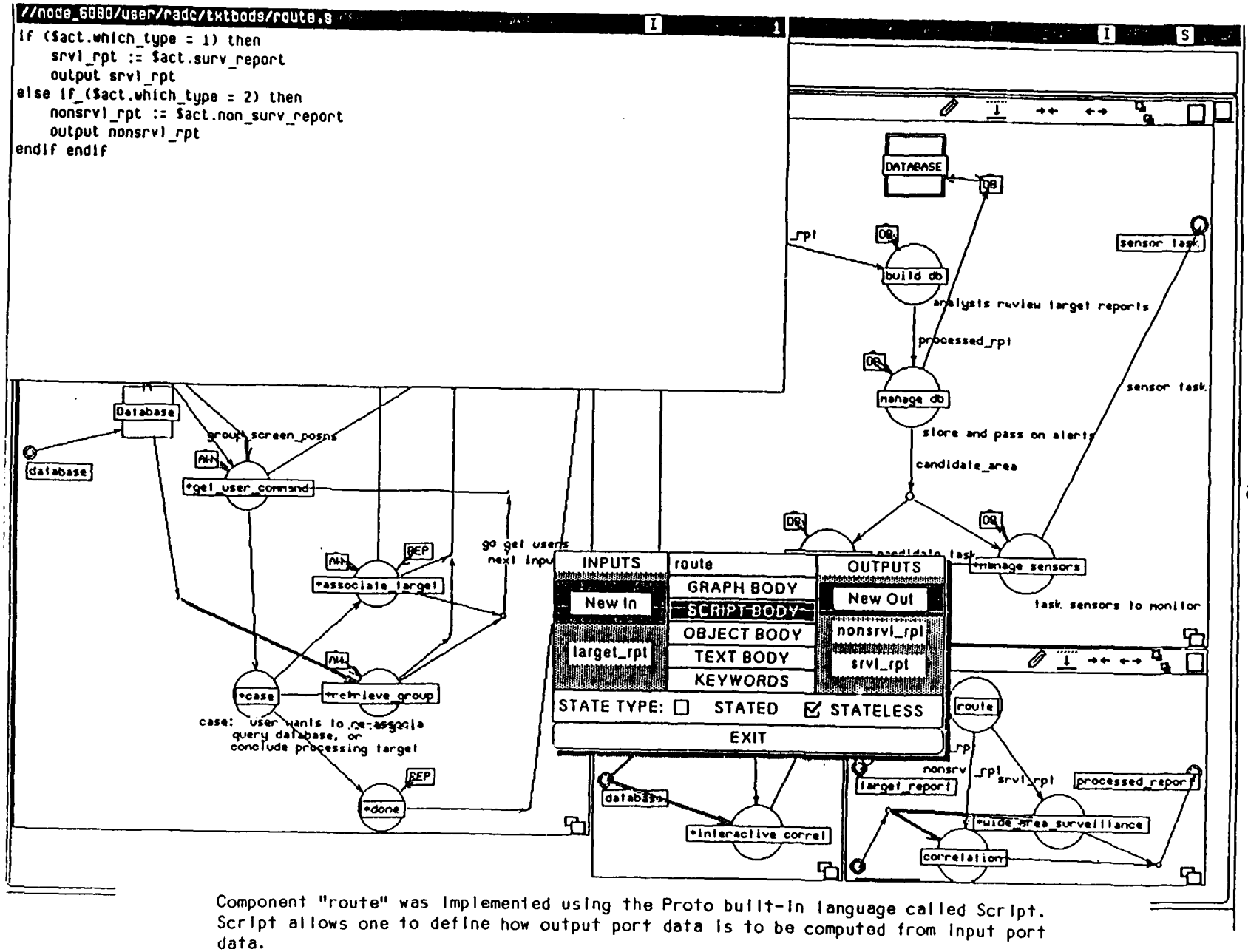
A component is selected by pointing device (mouse).
A pop-up menu lists the available component-specific actions:

GRAPH - brings up a window with component's dataflow definition.
OPEN - access to a components ports and multiple representations.
CONNECT - arc placement.
etc.

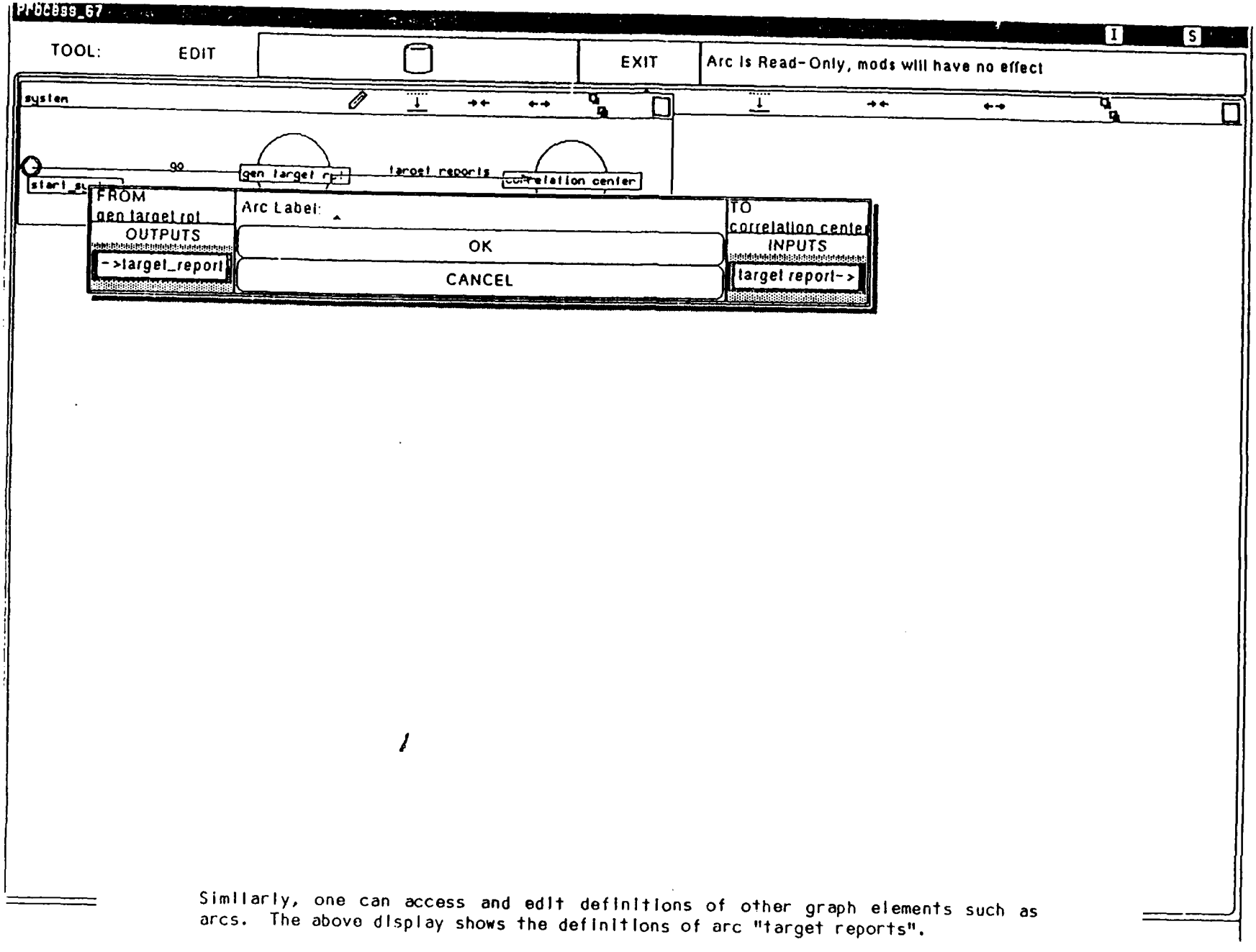
Selecting an action from the menu is the way to invoke that action.



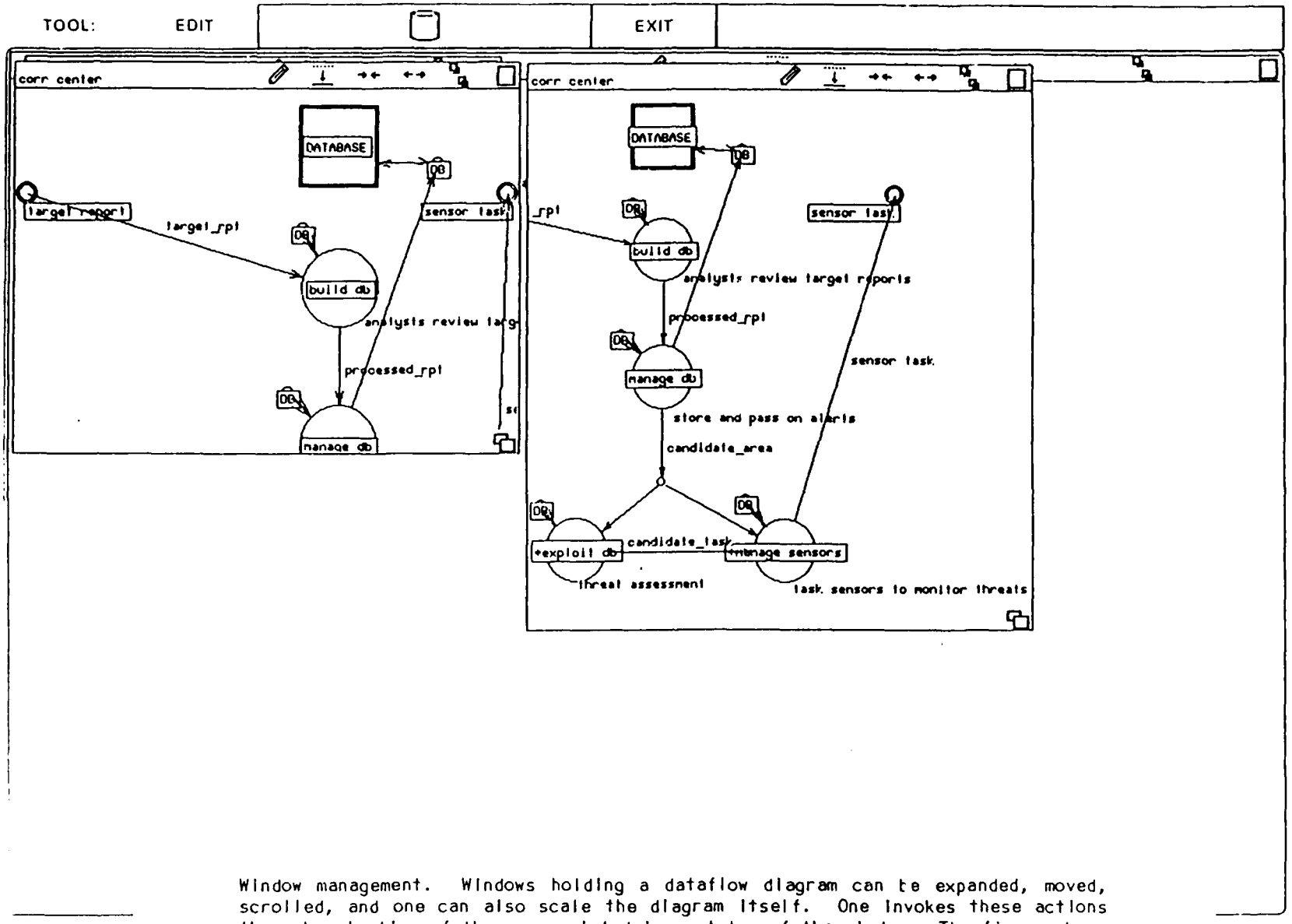
"OPEN" on component "route" shows its ports, and gives access to various types of definition for this component. Above, text body is selected, and it may be edited at this point with the standard Apollo screen editor.



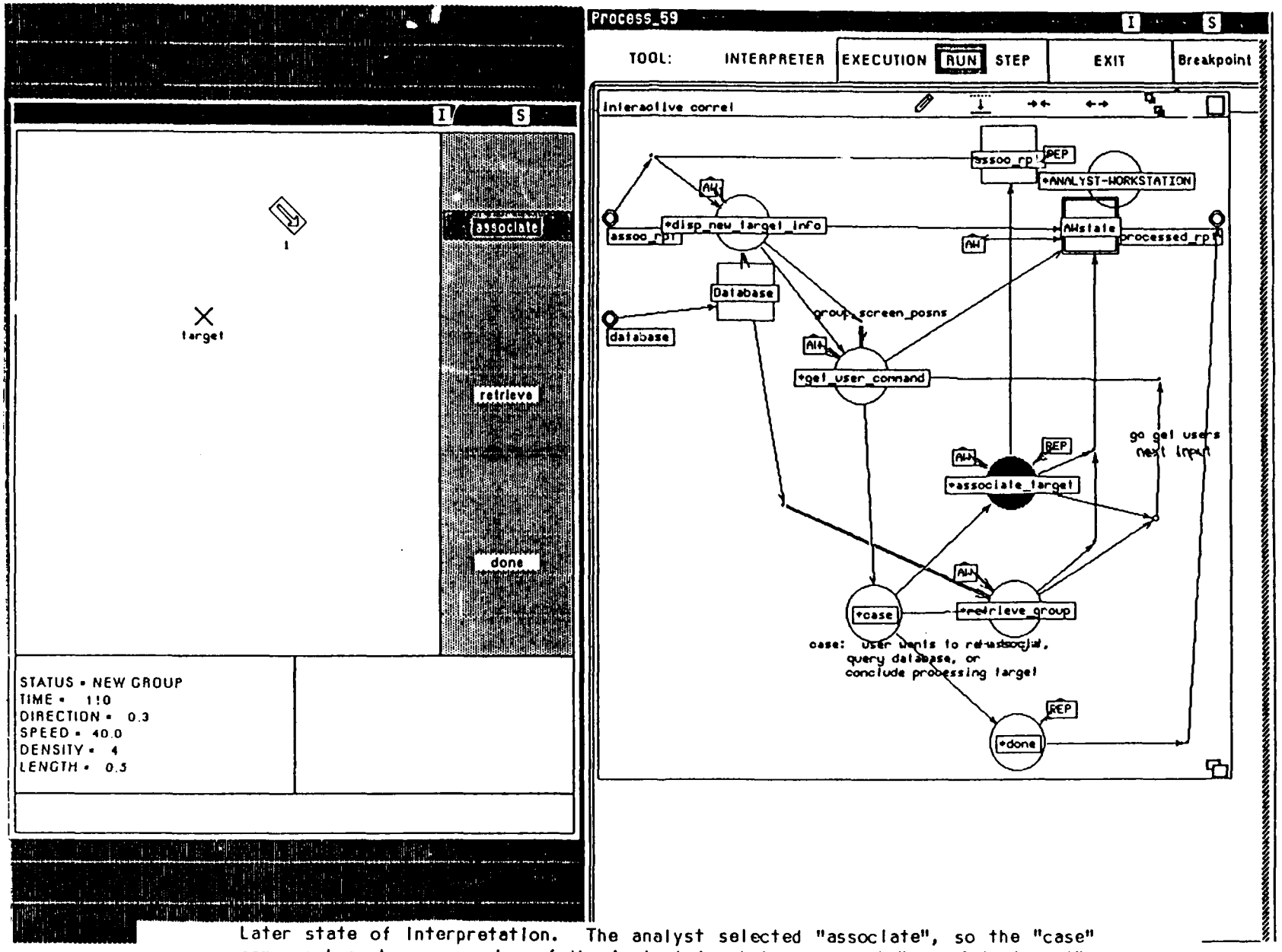
Component "route" was implemented using the Proto built-in language called Script. Script allows one to define how output port data is to be computed from input port data.



Similarly, one can access and edit definitions of other graph elements such as arcs. The above display shows the definitions of arc "target reports".



Window management. Windows holding a dataflow diagram can be expanded, moved, scrolled, and one can also scale the diagram itself. One invokes these actions through selection of the appropriated icon at top of the window. The figure above shows two windows holding the same dataflow description. The one on the right has been expanded, scrolled, and scaled.

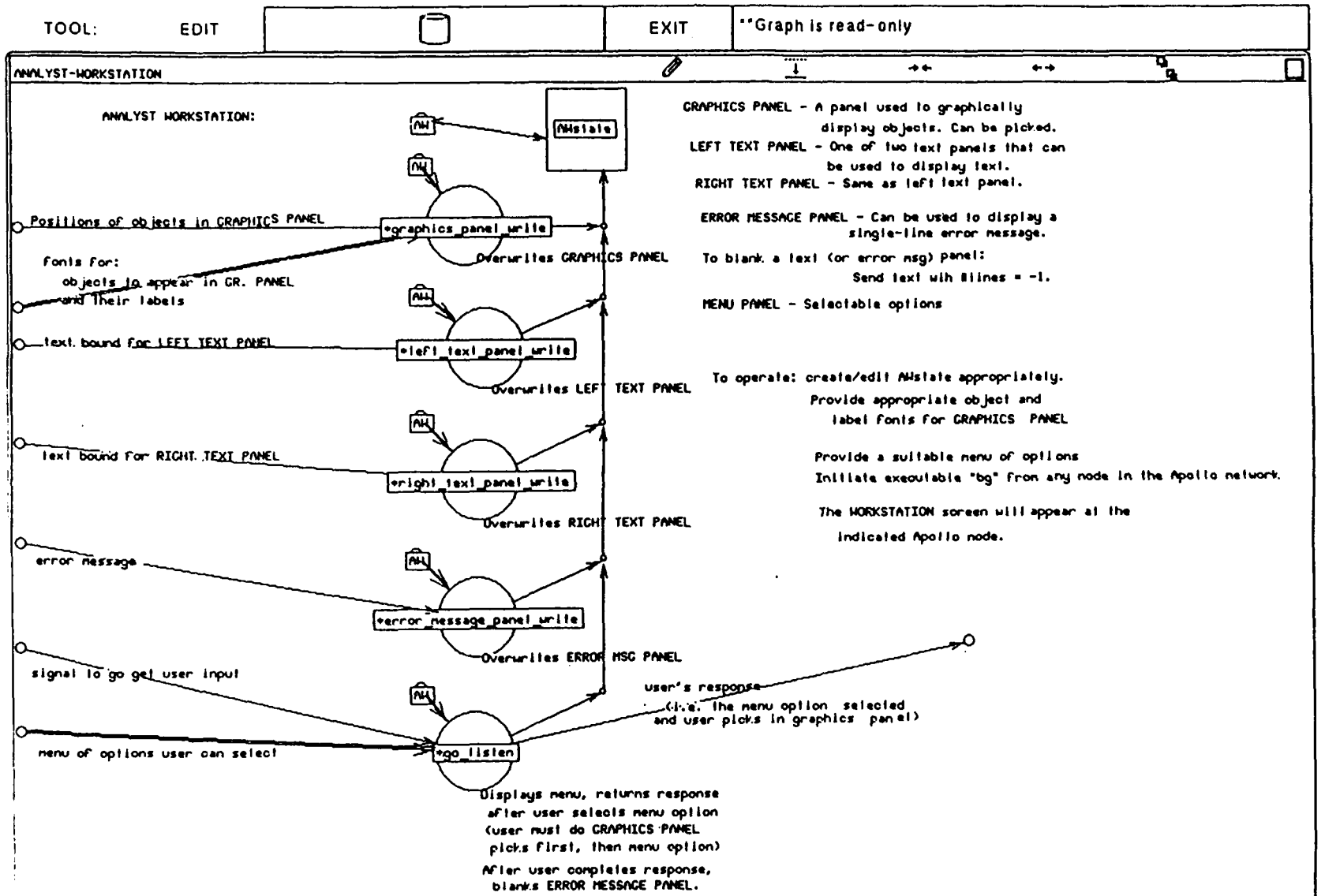


Later state of interpretation. The analyst selected "associate", so the "case" component routes processing of the Analyst input to component "associate target". In this manner, Analyst inputs are coupled to operations in the data flow.

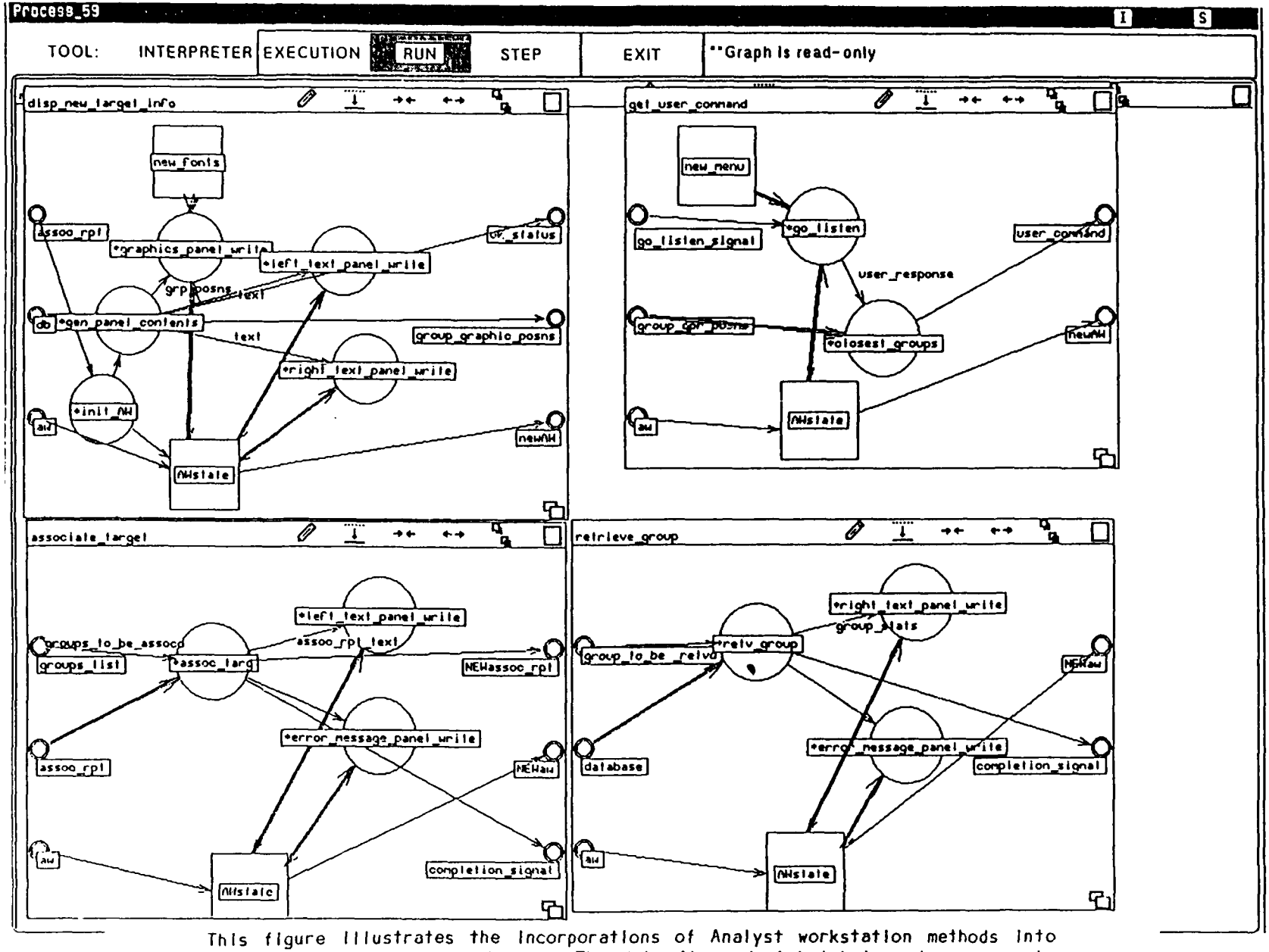
REUSE

Prototyping and reuse work well together. The availability of an application-specific library of reusable modules simplifies the construction of prototypes in that application area, and reduces the level of computing skills needed by the prototype specifier. Also, the normal impediments to reuse, i.e. performance and design robustness, are not critical parameters in prototyping. Prototyping encourages reuse because the system designer can observe the operation of a reusable modules in the target system context, and thereby gain better insight into the module's functions.

To facilitate reuse, a prototyping system needs database facilities which help in selection of candidate modules out of a reusable library. These facilities support classification of modules, searching for selected module properties, relation of modules to their datatypes, etc. In addition, the specification language and editing facilities should make it easy to insert (bind) a module into a specification. The following screens show the typical structure of a reusable module and its incorporation into a Proto specification.



This figure illustrates the Analyst Workstation object, exemplifying how a typical collection of reusable modules is documented in Proto. Methods are depicted as components. The message each method expects is indicated by an input arc. Presentation aspects that can be customized (e.g. the fonts and lists of menu options) are indicated as parameters. To use this object, the prototype definer copies the methods needed into the data flow being edited, indicating which workstation, fonts, and menu are being used by providing the appropriate input connections.



This figure illustrates the incorporations of Analyst workstation methods into low-level dataflow descriptions. The data flows depicted belong to components shown in the "interactive correlation" data flow (previously shown).

PROTO STRENGTHS

PROTO provides the ability to create system functionality specifications using dataflow diagrams, with dependence on a library of reusable modules to make prototype construction rapid. Direct enactment of those specifications via interpretation yields a functional prototype, by which those specifications can be reviewed against various input scenarios.

PROTO provides tools for building and viewing complex system descriptions, both in terms of hierarchical composition and in viewing multiple dimensions of description. It is a working prototype of a new architecture for software tools systems, where the object-oriented user interface is tightly coupled to an object-managed database. This permits multiple tools to share design data and screen images with consistent interpretations.